Solutions to Problems

from "Essentials of Electronic Testing"

©M. L. Bushnell and V. D. Agrawal, 2002

February 10, 2006

Please Read This

This manual contains solutions to all problems that appear at the end of the chapters in the book. At the end of the manual we have included the solutions to problems we used for the examinations in the Spring 2002 course at Rutgers University, and Spring 2004 and Spring 2005 courses at Auburn University.

In spite of all the care taken to ensure accuracy, we caution the user that some answers may contain errors as it is the first release of this manual. We will appreciate if any errors or comments are forwarded to us by email: *vagrawal@eng.auburn.edu* or *bushnell@caip.rutgers.edu*.

This manual has been created as teaching material that accompanies the book. To preserve its effectiveness, it should not be distributed. If necessary, only a very small set of solutions can be copied for distribution in the class. Please do not pass your copy on to others and ask any one requesting it to contact the authors.

Teachers can also use the presentation slides for 31 lectures (or an alternative sequence of 23-lectures), based on the book and available at the following websites:

http://www.eng.auburn.edu/∼vagrawal/COURSE/lectures.html

http://www.caip.rutgers.edu/∼bushnell/rutgers.html

We hope the readers of our book, both teachers and students, will benefit from this work. We acknowledge the help from colleagues and students in completing this solution manual and the assistance of the University of Wisconsin-Madison in its initial distribution.

Chapter 1: Introduction

1.1 Chip testing

The events of Example 1.1 are redefined as follows:

PQ:	chip is good	P:	chip passes the test
FQ:	chip is bad	F:	chip fails the test

A 70% yield means, $Prob(PQ) = 0.7$ and $Prob(FQ) = 0.3$. Following the analysis of Example 1.1, $Prob(P) = 0.68$. Then,

$$\begin{aligned} \text{Defect level} &= \frac{\text{Bad chips that pass tests}}{\text{All chips that pass tests}} \\ &= Prob(FQ|P) \\ &= \frac{Prob(P|FQ)Prob(FQ)}{Prob(P)} \\ &= \frac{0.05 \times 0.3}{0.68} = 0.022 \end{aligned}$$

The defect level is 22,000 ppm (parts per million).

1.2 Chip testing

Let x denote the escape probability, $Prob(P|FQ)$. Referring to the formula derived in Problem 1.1, a defect level of 500 *ppm* means,

$$\frac{Prob(P|FQ)Prob(FQ)}{Prob(P)} = \frac{x \times 0.3}{0.95 \times 0.7 + x \times 0.3} = 0.0005$$

This gives,

$$x = \frac{0.0003325}{0.29985}$$

Next, we obtain,

$$\begin{aligned} \text{Defect coverage} &= Prob(F|FQ) = 1 - Prob(P|FQ) \\ &= 1 - x = 0.99889 \end{aligned}$$

The required defect coverage is 99.889%. This represents the capability of the test in detecting the actual "defects" that occur and should not be confused with the "fault coverage," which is defined for the "single stuck-at" fault model.

1.3 Test cost

Assuming that one vector is applied per clock cycle during the digital test, the rate of test application is 200 million vectors per second. Therefore,

$$\text{Digital test time} = \frac{1000 \times 10^6}{200 \times 10^6} = 5 \ s$$

Adding the analog test time, we get

$$\text{Total test time} = 1.5 + 5.0 = 6.5\ s$$

The testing cost for a 500 MHz, 1,024 pin tester was obtained as 4.56 *cents* in Example 1.2 (see page 11 of the book.) Thus,

$$\text{Cost of testing a chip} = 6.5 \times 4.56 = 29.64\ cents$$

The cost of testing bad chips should also be recovered from the price of good chips. Since the yield of good chips is 70%, we obtain

$$\text{Test cost in the price of a chip} = \frac{29.64}{0.7} \approx 42\ cents$$

41.8 cents should be included as the cost of testing while figuring out the price of chips.

1.4 Test cost and self-test

Following Example 1.2 of the book (pp. 10-11), we obtain

$$\text{ATE purchase price} = \$1.2M + 256 \times \$3,000 = \$1.968M$$

Assuming a 20% per year linear rate of depreciation, a maintenance cost of 2% of the price, and an annual operating cost of $\$0.5M$,

$$\text{Running cost} = \$1.968M \times 0.2 + \$1.968M \times 0.02 + \$0.5M = \$932,960/year$$

$$\text{Testing cost} = \frac{\$932,960}{365 \times 24 \times 3600} = 2.96\ cents/second$$

Testing cost of the self-test design is 2.96 cents per second, down from 4.56 cents per second calculated in Example 1.2

1.5 Test complexity

Consider a cube of side d. The number of transistors (N_t) is proportional to the volume d^3, and the number of pins (N_p) is proportional to the surface area $6d^2$. Thus, the Rent's rule for the cube can be expressed as,

$$N_p = K \times {N_t}^{2/3}$$

where K is a constant, which depends on such technology parameters as the minimum feature spacing. For simplicity, we will assume that this constant is the same for the flat and cubic chips. Following Example 1.3 (pp. 12-13 of book), we define the test complexity, TC, as transistors per pin, or $TC = N_t/N_p$. For the cube,

$$TC_{cube} = \frac{N_t}{N_p} = \frac{N_t}{K{N_t}^{2/3}} = \frac{1}{K}{N_t}^{1/3}$$

Using the Rent's rule for a flat chip (Equation 1.5 on page 13 of book), we obtain

$$TC_{square} = \frac{N_t}{K{N_t}^{1/2}} = \frac{1}{K}{N_t}^{1/2}$$

Therefore,

$$\frac{TC_{square}}{TC_{cube}} = {N_t}^{1/6}$$

This ratio of test complexities continues to increase as the number of transistors (N_t) on the VLSI device grows. For example, for $N_t = 1$ *million*, the square-chip test complexity is ten times greater than that of the cubic-device. **The test problem of the cubic configuration is less complex than that for the flat chip.**

Note: Although chips at present are not designed as three-dimensional objects, three-dimensional packages and interconnects are in use. An interested reader may see the article: H. Goldstein, "Packages Go Vertical," *IEEE Spectrum*, vol. 38, no. 8, pp. 46-51, August 2001. Recently, Matrix Semiconductor announced plans to produce a three-dimensional memory chip. See, "Adding a Third Dimension to Chips," *Computer*, vol. 35, no. 3, p. 29, March 2002.

Chapter 2: VLSI Testing Process and Test Equipment

2.1 Test types

To reduce the warranty and product liability costs, the manufacturer must adopt a thorough but cost-effective test plan. A low failure rate, which may be as low as 100 parts per million, means that among one million chips shipped by the manufacturer there should be no more than 100 defective chips. A suitable test strategy requires adjustments to tests as the production ramps up. A realistic plan is as follows:

- Initial production: The manufacturer uses parametric tests and vector tests, the latter with coverage in the 95-100% stuck-at fault range. For high-speed microprocessor chips, at-speed critical path tests are run. The chips should be subjected to burn-in test for infant mortality.
- Matured production: If burn-in failures are lower than the required defect level then that test is eliminated or reduced to a sample basis. Any field returns are re-tested by the manufacturing tests. If these pass then the manufacturing tests are augmented, when necessary, by customer-supplied tests.
- Test optimization: Tests are optimized to reduce the manufacturing cost. First, test sequences that fail a larger number of devices are moved to the beginning. Second, test sequences that do not fail any devices are dropped. Such modifications change the emphasis from detection of *modeled faults* to detection of actual *defects*.
- Process monitoring: Once the chip goes into high-volume production, the manufacturing process and the outgoing product (chips) should be monitored to keep any variations within *statistical limits*. This means that various parameters, such as metal resistivity, polysilicon conductivity, transistor parameters, etc., should be within their three-sigma range ($average \pm 3 \times standard\ deviation$). Any excursions outside such a range are immediately diagnosed and the causes remedied.

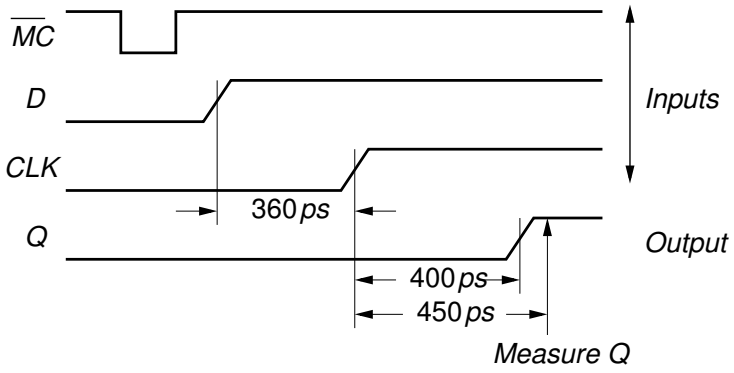
2.2 Contact test

Assume a diode drop of $0.7V$. Then, the pin voltage range for contact test is given by:

$$\begin{aligned} \text{Upper range}: \ V_{pin} &= 0V - 0.7V - 100\mu A \times 2000\Omega \\ &= -0.9V \\ \text{Lower range}: \ V_{pin} &= 0V - 0.7V - 250\mu A \times 2000\Omega \\ &= -1.2V \end{aligned}$$

2.3 Set-up time test

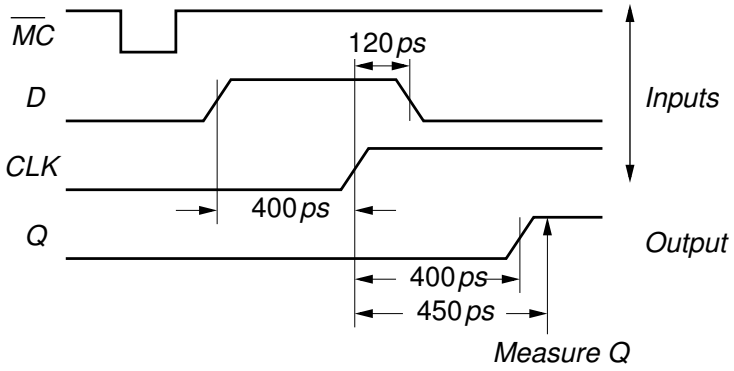
To test a set-up time, $t_{set-up} = 360ps$, apply the following waveforms to the chip (a clock-to-Q delay of $400ps$ is assumed):



At an interval of $450ps$ after the rising CLK edge, measure Q on the ATE. If $Q = 1$, the device passes, otherwise it fails. Using $\overline{MS}$ instead of $\overline{MC}$, repeat the above waveform sequence, but with D inverted and the expected Q signal also inverted. At an interval of $450\mu s$ after the rising CLK edge, again measure Q on the ATE. If $Q = 0$, the device passes, otherwise it fails. The same waveforms are applied simultaneously to all five D lines, and five simultaneous measurements are made on the five Q lines.

2.4 Hold time test

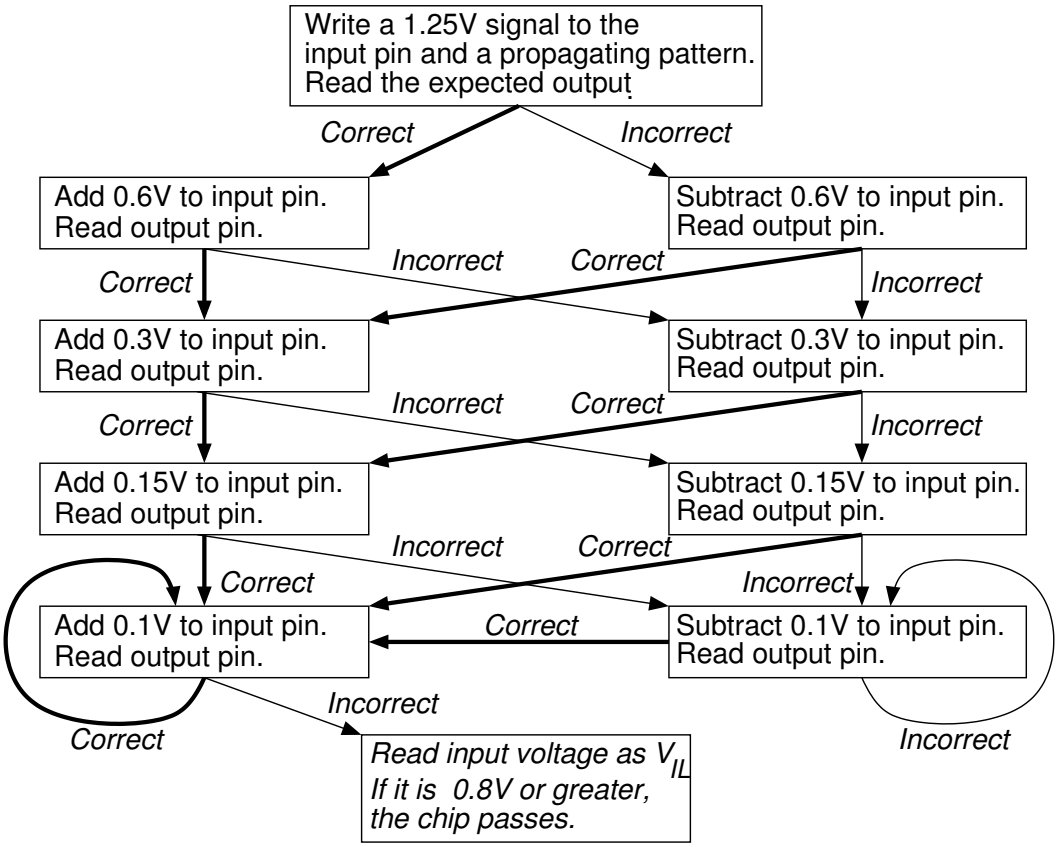
To test a hold time, $t_{hold} = 120ps$, apply the following waveforms to the chip (a clock-to-Q delay of $400ps$ is assumed):



At an interval of $120ps$ after the rising CLK edge, we lower the D line. If $Q = 1$ $450ps$ after the rising CLK edge, the device passes, otherwise it fails. Using $\overline{MS}$ instead of $\overline{MC}$, repeat the above waveform sequence, but with D inverted and the expected Q signal also inverted. At an interval of $450\mu s$ after the rising CLK edge, again measure Q on the ATE. If $Q = 0$, the device passes, otherwise it fails. The same waveforms are applied simultaneously to all five D lines, and five simultaneous measurements are made on the five Q lines.

2.5 Threshold test

Perform the threshold test as given on page 32 of the book, but with the following changes: Assume a $5V$ supply, and perform binary search to find V_{IL} and V_{IH}. The following procedure determines V_{IL}:



The advantage of this procedure is that it greatly speeds up the test. The test for V_{IH} is analogous.

Chapter 3: Test Economics and Product Quality

3.1 Economic decision

We start with the following formula for the price of the car deriven by John (Equation 3.2 on page 38 of the book):

$$P = 20,000 + \frac{20,000}{n} \quad dollars$$

where n is the number of breakdowns per 15,000 miles since John's car is driven 15,000 miles in a year. Because Laura drives only 5,000 miles per year, her car is expected to have $n/3$ breakdowns per year. Assuming a linear depreciation to zero value over 20 years and an average repair cost of \$250 per breakdown, the annual cost of driving is

$$\begin{aligned} C &= \frac{P}{20} + K + 250n/3 \quad dollars \\ &= 1,000 + \frac{1,000}{n} + K + 250n/3 \quad dollars \end{aligned}$$

where K is the cost of gasoline and regular maintenance, assumed to be the same for all models. To minimize this cost, we write

$$\frac{dC}{dn} = -\frac{1,000}{n^2} + \frac{250}{3} = 0 \quad \text{or} \quad n = \sqrt{12}$$

This is a minimum because $\frac{d^2C}{dn^2} > 0$. The price of a car for minimum transportation cost is,

$$P = 20,000 + \frac{20,000}{\sqrt{12}} = 25,774 \; dollars$$

Laura should invest in a car priced around 25,774 dollars.

3.2 Economic decision

(a) Let x be the daily wages of a technician and c be the cost of components on a board. When n technicians work in the assembly shop, the cost of one board is,

$$\begin{aligned} C(n) &= \frac{Warehouse\ cost}{n} + technician's\ wages + component\ cost \\ &\quad + workspace\ cost \\ &= \frac{10,000}{n} + x + c + \frac{500n^2}{n} \end{aligned}$$

To minimize this cost, we write

$$\frac{dC(n)}{dn} = -\frac{10,000}{n^2} + 1,000n = 0 \quad \text{or} \quad n = \sqrt{20} = 4.47$$

This is a minimum since $\frac{d^2C(n)}{dn^2} > 0$. We obtain the minimum cost as,

$$C(4) = C(5) = \$4,500 + x + c$$

To minimize the cost we should either hire four technicians, or reduce the workforce to five if more than five technicians were already employed.

(b) Substituting $x = 200$ and $c = 10,000$ in the last equation, we get

$$C(4 \text{ or } 5) = \$4,500 + 200 + 10,000 = \$14,700$$

The minimum cost of a single-board system is $14,700.

3.3 Benefit-cost analysis

Please note a correction in the statement of this problem. The part (a) should read: Show that this scheme is beneficial for chips whose total cost is less than ten times the burn-in cost when the burn-in yield is 90%.

(a) *Complete elimination of burn-in:* Let C_t be the total cost of a chip in the present scheme where burn-in test is applied to every chip that passes the conventional test. Let C_b be the per chip cost of burn-in. C_t includes C_b, as well as another component, C_f, which accounts for the costs of fabrication, conventional test, etc. It is given by,

$$C_t = \frac{C_f + y_c C_b}{y_c y_b}$$

where y_c is the yield with the conventional test and y_b is the yield reduction due to burn-in. Since the cost of I_{DDQ} test is 10% of the burn-in cost and there is a 10% yield loss, the cost of a chip when burn-in is replaced by I_{DDQ} test is given by,

$$C'_t = \frac{C_f + 0.1 y_c C_b}{0.9 y_c y_b}$$

For the new scheme to be beneficial, we must have

$$C'_t < C_t \quad \text{or} \quad C_t < \frac{9C_b}{y_b}$$

For the given 90% burn-in yield, $y_b = 0.9$, and $C_t < 10C_b$. **The total cost should not exceed ten times the burn-in cost.**

(b) *Apply burn-in test only to chips that fail I_{DDQ} test:* Let y_b be the burn-in yield. Consider all chips that have passed pre-burn-in tests. A fraction y_b of these is "good" chips. We apply I_{DDQ} test to all chips passing the pre-burn-in test. Due to the 10% yield loss, this will produce a fraction $0.9y_b$ consisting of good chips. The remaining fraction, $1 - 0.9y_b$, must be subjected to the burn-in test to recover the lost yield. For the new scheme to be beneficial, we must have

$$0.1C_b + (1 - 0.9y_b)C_b < C_b \quad \text{or} \quad y_b > \frac{1}{9}$$

Burn-in yield should be greater than 1/9 or 11.1%.

3.4 Yield and cost

Let C_w be the cost of processing a wafer having N chips and let $y(A)$ be the yield of chips, where A is the chip area. Then the cost per good chip is obtained as,

$$C_c = \frac{C_w}{Ny(A)}$$

DFT changes the chip area to $(1+\Delta)A$. The number of chips on a wafer of area NA is now given by, $NA/(A+\Delta A) = N/(1+\Delta)$. The cost of a good chip with DFT is given by,

$$C_c(DFT) = \frac{C_w}{\frac{N}{1+\Delta}\ y(A+\Delta A)}$$

Therefore, the cost increase due to DFT is,

$$\begin{aligned} \text{Cost increase} &= \frac{C_c(DFT) - C_c}{C_c} \times 100 \ \ percent \\ &= \left[\frac{(1+\Delta)y(A)}{y(A+\Delta A)} - 1\right] \times 100 \ \ percent \end{aligned}$$

Using the yield formula of Equation 3.12 (p. 46 in the book), we get

$$\begin{aligned} \text{Cost increase} &= \left[(1+\Delta)\frac{(1+Ad/\alpha)^{-\alpha}}{(1+(1+\Delta)Ad/\alpha)^{-\alpha}}\right] \times \ \ percent \\ &= \left[(1+\Delta)\left(1+\frac{Ad\Delta}{\alpha+Ad}\right)^{\alpha} - 1\right] \times 100 \ \ percent \end{aligned}$$

which is the required result.

For the given data, $d = 1.25\ defects/cm^2$, $\alpha = 0.5$, $\Delta = 0.1$, and $A = 1\ cm^2$, we obtain

$$\begin{aligned} \text{Cost increase} &= \left[1.1\left(1+\frac{1.25\times 0.1}{0.5\times 1.25}\right)^{0.5} - 1\right] \times 100 \ \ percent \\ &= 13.86\% \end{aligned}$$

There is a 13.86% increase in the chip cost due to DFT.

3.5 Defect level and fault coverage

Defect level, DL, is given by Equation 3.20 (p. 50 of the book), as follows:

$$DL = 1 - \left(\frac{\beta + TAf}{\beta + Af}\right)^{\beta}$$

where T is the fault coverage, Af is the average number of faults on a chip of area A, and β is a fault clustering parameter. Further manipulation of this equation

leads to the following result:

$$\begin{aligned}(1-DL)^{1/\beta} &= \frac{\beta+TAf}{\beta+Af} \\ \text{or} \qquad T &= \frac{(\beta+Af)(1-DL)^{1/\beta}-\beta}{Af}\times 100 \;\; percent\end{aligned}$$

which is the required result.

3.6 Defect level and fault coverage

Substituting the given fault density, $f = 1.45\ faults/cm^2$, the fault clustering parameter, $\beta = 0.11$, and the fault coverage, $T = 0.95$, in Equation 3.20 (page 50 of the book), we obtain the defect level as,

$$\begin{aligned}DL(T) &= 1-\left(\frac{\beta+TAf}{\beta+Af}\right)^{\beta} \\ &= 1-\left(\frac{0.11+0.95\times 1.0\times 1.45}{0.11+1.0\times 1.45}\right)^{0.11} \\ &= 0.00522 \;\text{ or }\; 5,220 \text{ parts per million}\end{aligned}$$

The defect level is 5,220 parts per million (ppm).

(a) To obtain the fault coverage T for a required defect level of 1,000 ppm, we substitute $DL = 0.001$ in the formula derived in Problem 3.5. Thus,

$$T = \frac{(0.11+1.45)\times 0.999^{1/0.11}-0.11}{1.45}\times 100 = 0.990$$

The required fault coverage is 99%.

(b) For a defect level of 500 ppm ($DL = 0.0005$), we get

$$T = \frac{(0.11+1.45)\times 0.9995^{1/0.11}-0.11}{1.45}\times 100 = 0.995$$

The required fault coverage is 99.5%.

3.7 Defect level

Defect level, $DL(T)$, given by Equation 3.20 (p. 50 of the book), can be written as:

$$\begin{aligned}DL(T) &= 1-\frac{(1+TAf/\beta)^{\beta}}{(1+Af/\beta)^{\beta}} \\ &= 1-\frac{e^{TAf}}{e^{Af}} = 1-e^{-Af(1-T)}, \;\; \text{as} \;\; \beta\to\infty\end{aligned}$$

Also, as $\beta \to \infty$, Equation 3.19 (p. 50 of the book) gives the yield,

$$Y = \left(1+\frac{Af}{\beta}\right)^{-\beta} = e^{-Af}$$

Substituting this expression for yield in the defect level, we get

$$DL(T) = 1 - (e^{-Af})^{1-T} = 1 - Y^{1-T}$$

which is the required result.

Chapter 4: Fault Modeling

4.1 Boolean functions

An n-variable Boolean function is completely specified by its truth-table. The output column in this table is a 2^n-bit vector that can be in 2^{2^n} distinct states, each specifying a different Boolean function.

4.2 Initialization faults

In the circuit of Figure 4.1 (p. 62 of the book), let Q_p denote the present state at the output of the FF. Let the next state, i.e., the output of the AND gate, be Q_n. We can write the *next state function*, as

$$Q_n = (Q_p + A)(\overline{A} + B)$$

If we set $A = 1$, the next state function, $Q_n = B$, becomes independent of the present state. That is, irrespective of the present state, the next state can be set to a value, which is uniquely determined by primary inputs. This makes the fault-free circuit initializable. When the fault A s-a-0 is present, the above equation reduces to $Q_n = Q_p$. Thus, starting with $Q_p = X$, Q_n can never be changed to any value other than X and, therefore, **the circuit will remain uninitialized in the presence of this fault.**

Using the next-state expression, we can easily determine that no other single stuck-at fault in this circuit will prevent initialization. For example, consider the s-a-0 fault on the top branch of the fanout of A. The faulty next state function is $Q_n = Q_p(\overline{A} + B)$, which can be set to 0, when $Q_p = X$, by applying $A = 1,\ B = 0$.

4.3 Fault counting

See Section 4.5 (last paragraph on p. 70 of the book.)

4.4 Fault counting

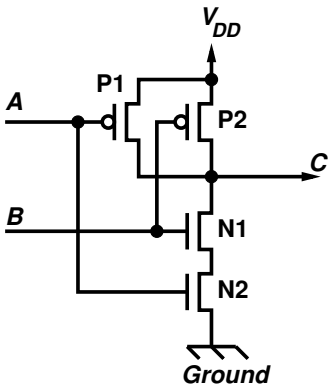
For the circuit of Figure 4.6 (p. 72 of book), we have

$$\begin{aligned} \text{Number of fault sites} &= \text{PIs} + \text{gates} + \text{fanout branches} \\ &= 2 + 4 + 6 = 12 \end{aligned}$$

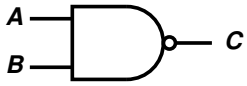
Therefore,

$$\begin{aligned} \text{Number of single and multiple faults} &= 3^{\text{number of fault sites}} - 1 \\ &= 3^{12} - 1 = 531,440 \end{aligned}$$

The circuit has 531,440 single and multiple stuck-at faults.



CMOS NAND gate.



Logic NAND gate.

Circuit for Problem 4.5.

4.5 CMOS faults

(a) A two-input NAND gate is shown in the above figure. The following table gives tests for transistor stuck-open (sop) faults:

Test No.	Fault	Test: Vector 1, Vector 2
1	P1 sop	11, 01
2	P2 sop	11, 10
3	N1 sop	01, 11 or 10, 11 or 00, 11
4	N2 sop	01, 11 or 10, 11 or 00, 11

Notice that the sop faults of N1 and N2 have exactly the same tests. These two faults are equivalent. Equivalence of transistor faults is discussed in the following paper:
M.-L Flottes, C. Landrault and S. Provossoudovitch, "Fault Modeling and Fault Equivalence in CMOS Technology," *J. Electronic Testing: Theory and Applications*, vol. 2, pp. 229-241, August 1991.
(b) The following sequence of four vectors contains one vector pair for each fault in the above table:

$$11,\ 01,\ 11,\ 10$$

Notice that this sequence also detects all single stuck-at faults in the logic model of the NAND gate.
(c) A stuck-at fault in a signal affects two transistors in the two-input NAND gate. For example, the fault A s-a-1 will mean that N1 remains permanently shorted (N1-ssh) and P1 remains permanently open (P1-sop). The following table gives all equivalences:

Stuck-at fault	Equivalent transistor faults
A s-a-1	N1-ssh and P1-sop
B s-a-1	N1-ssh and P2-sop
C s-a-1	(P1-ssh or P2-ssh) and (N1-sop or N2-sop)
A s-a-0	N1-sop and P1-ssh
B s-a-0	N2-sop and P2-ssh
C s-a-0	N1-ssh, N2-ssh, P1-sop and P2-sop

Notice that the three equivalent faults, A s-a-0, B s-a-0 and C s-a-0, are actually caused by different faulty transistors. They are detected by the same test (11).

4.6 Fault models

See Section 4.4 in the book.

4.7 Fault indistinguishability

Without loss of information we will write a function $f(V)$ as f. Thus, the left hand side of Equation 4.3 is:

$$
\begin{aligned}
&[f_0 \oplus f_1] \oplus [f_0 \oplus f_2] \\
&= [f_0\overline{f_1} + \overline{f_0}f_1] \oplus [f_0\overline{f_2} + \overline{f_0}f_2] \\
&= (f_0\overline{f_1} + \overline{f_0}f_1)\overline{(f_0\overline{f_2} + \overline{f_0}f_2)} + \overline{(f_0\overline{f_1} + \overline{f_0}f_1)}(f_0\overline{f_2} + \overline{f_0}f_2) \\
&= (f_0\overline{f_1} + \overline{f_0}f_1)\overline{(f_0\overline{f_2})}\,\overline{(\overline{f_0}f_2)} + \overline{(f_0\overline{f_1})}\,\overline{(\overline{f_0}f_1)}(f_0\overline{f_2} + \overline{f_0}f_2) \\
&= (f_0\overline{f_1} + \overline{f_0}f_1)(\overline{f_0} + f_2)(f_0 + \overline{f_2}) + (\overline{f_0} + f_1)(f_0 + \overline{f_1})(f_0\overline{f_2} + \overline{f_0}f_2) \\
&= (f_0\overline{f_1} + \overline{f_0}f_1)(\overline{f_0}\ \overline{f_2} + f_0f_2) + (\overline{f_0}\ \overline{f_1} + f_0f_1)(f_0\overline{f_2} + \overline{f_0}f_2) \\
&= f_0\overline{f_1}f_2 + \overline{f_0}f_1\overline{f_2} + \overline{f_0}\ \overline{f_1}f_2 + f_0f_1\overline{f_2} \\
&= (\overline{f_1}f_2)(f_0 + \overline{f_0}) + f_1\overline{f_2}(\overline{f_0} + f_0) \\
&= \overline{f_1}f_2 + f_1\overline{f_2} \\
&= f_1 \oplus f_2 \\
&= \text{Left hand side of Equation 4.4}
\end{aligned}
$$

This completes the derivation of Equation 4.4 from Equation 4.3.

4.8 Functional equivalence

Faulty functions for the circuit of Figure 4.12 corresponding to the two faults are:

$$
\begin{aligned}
i(c\ s-a-0) &= b(\overline{ab}) = \overline{a}b \\
i(f\ s-a-1) &= (a+b)\overline{a} = \overline{a}b
\end{aligned}
$$

The two faulty functions are indistinguishable and hence **the two faults are equivalent.**

4.9 Functional equivalence

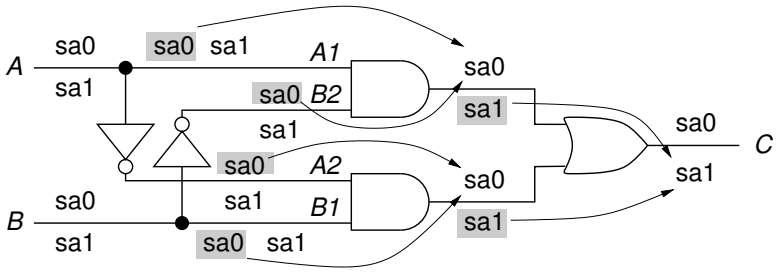
Faulty functions for the circuit of Figure 4.6 corresponding to the two faults are:

$$\begin{aligned} z(c\ s-a-1) &= \overline{\overline{\overline{ab}.(\overline{\overline{ab}.b})}} \\ &= \overline{ab.(ab+\overline{b})} = \overline{ab} \\ z(f\ s-a-1) &= \overline{ab} \end{aligned}$$

The two faulty functions are indistinguishable and hence **the faults are equivalent**.

4.10 Fault collapsing for test generation

The circuit of Figure 4.9 has 18 single stuck-at faults. Gate-level fault equivalence, as shown in the following figure, reduces the number to 12. The faults in shaded boxes have been collapsed as shown by arrows. Many ATPG and fault simulation



programs will collapse faults as shown above. However, functional fault collapsing can further reduce the number of faults to 10. As shown in Example 4.11 (see page 75 of the book), the s-a-1 faults on $A1$ and $B1$ are equivalent, and so are the s-a-1 faults on $A2$ and $B2$.

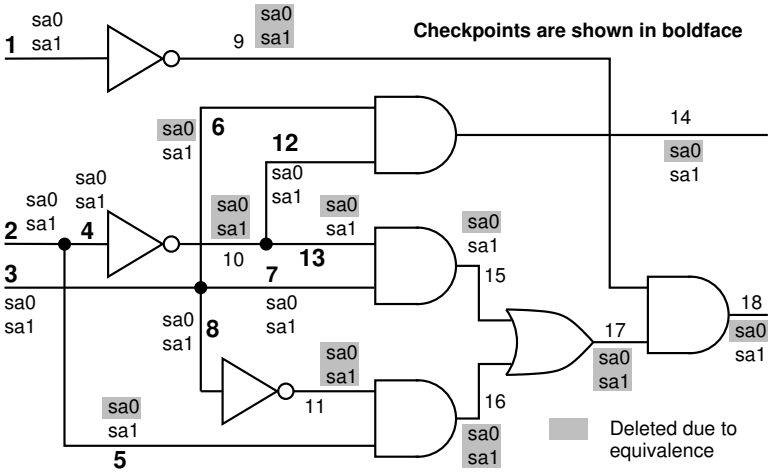
Whether we take the set of 12 faults or the set of 10 faults, their detection requires all four input vectors.

4.11 Equivalence and dominance fault collapsing

(a) The given circuit is shown below with fault sites marked by numbers. The number of potential fault sites is 18. The total number of faults is 36.

(b) The figure shows deletion of equivalent faults using an output to input pass. Of the 36 faults, 20 remain, giving a collapse ratio $20/36 = 0.56$.

(c) Checkpoint lines are shown by boldface numbers. These are three PIs and seven fanout branches. Line 2 fans out to 4 and 5. Line 3 fans out to 6, 7 and 8. Line 10 fans out to 12 and 13. There are ten checkpoints and 20 checkpoint faults. Further, s-a-0 faults on lines 6 and 12 are equivalent and any one of them can be chosen. Similarly, s-a-0 faults on 7 and 13 are equivalent, and so are s-a-0 on 5 and s-a-1 on 8. Thus, the size of the fault set is reduced to 17, giving a collapse ratio $17/36 = 0.47$.

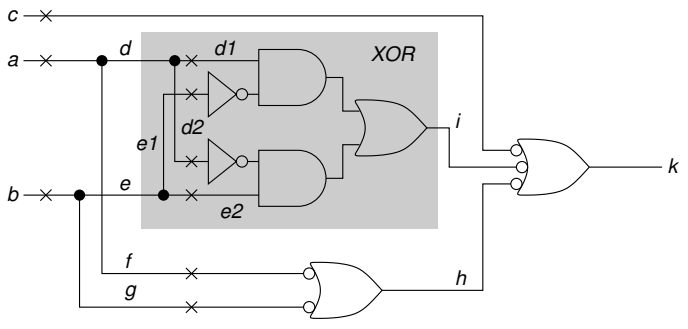


Circuit for Problem 4.11: (b) Equivalence collapse ratio = 20/36 = 0.56
(c) Dominance (uncollapsed faults at checkpoints) collapse ratio = 17/36 = 0.47

4.12 Dominance fault collapsing

(a) Checkpoints are defined for the signals in a combinational circuit. These signals are the interconnects between Boolean gates, a fact not always explicitly stated. To avoid ambiguity, the definition on page 78 of the book should read as:
Definition 4.7 *Checkpoints. Primary inputs and fanout branches of a combinational circuit consisting only of Boolean gates are called the* checkpoints.

To find checkpoints of the circuit of Figure 4.12, we must replace the exclusive-OR (XOR) function by a primitive Boolean gate implementation. AND, OR, NAND, NOR and NOT are called the *primitive* Boolean gates. Functions such as XOR are sometimes referred to as *complex* gates. In the following figure, we have assumed one such implementation. Our result is, therefore, based on this assumption. Other implementations of the XOR function are possible and can give a different set of checkpoints.



There are nine checkpoints in this circuit. These include three primary inputs, a, b and c, and six fanout branches, $d1$, $d2$, f, $e1$, $e2$ and g. The checkpoint fault set consists of eighteen faults – s-a-0 and s-a-1 faults on the nine lines.

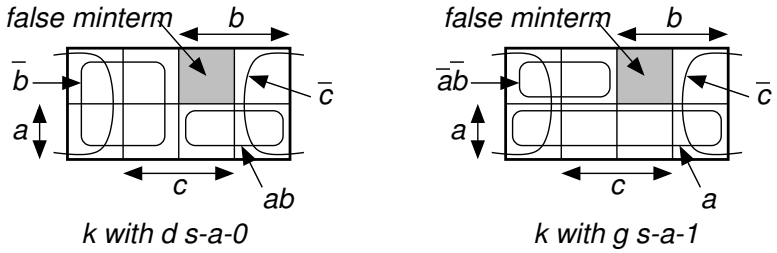
Notice that lines d and e of the original circuit are not checkpoints. If we did

not model the XOR block with Boolean gates, then those lines will appear to be checkpoints, whose number will be fourteen. However, detection of those faults will not guarantee detection of faults on the fanouts that are internal to the XOR block. Considering the Boolean gate structure, a fault on d corresponds to a simultaneous (multiple) fault on $d1$ and $d2$ and, in general, the detection of a multiple fault is not equivalent to detection of the component faults.

(b) We evaluate the output function k corresponding to the two faults:

$$\begin{aligned} k(d\ s-a-0) &= \overline{c}+\overline{b}+\overline{\overline{a}+\overline{b}} \\ &= \overline{c}+\overline{b}+ab \\ k(g\ s-a-1) &= \overline{c}+\overline{a\overline{b}+\overline{a}b}+a \\ &= \overline{c}+\overline{a}\overline{b}+a \end{aligned}$$

The two faulty functions are shown by Karnaugh maps below. In both cases, the functions have exactly one false minterm, $\overline{a}bc$. **Since the two faulty functions are identical the corresponding faults are equivalent**.



Note: this type of fault equivalence is functional and is often difficult to find by typical fault analysis tools, which rely on structurally identifiable equivalences.

Chapter 5: Logic and Fault Simulation

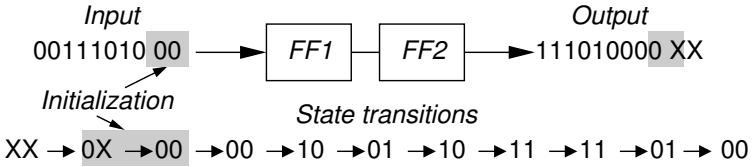
5.1

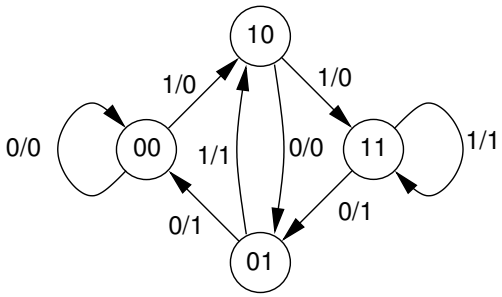
The eight vectors of Table 5.1 apply all possible inputs to each stage of the 4-bit ripple-carry adder. The longest path is from the carry input C_0 to carry output C_4. The delay of this path is tested by two vector pairs. Vector 2 followed by vector 6 applies a rising transition at C_0, which ripples through the circuit to the C_4 output. Similarly, vector 7 followed by vector 3, propagates a falling transition. The rearranged vectors are given in the following table. This vector set still applies all input states to all stages of the adder and hence is a better verification sequence.

Vec. no.	$C_0A_0B_0A_1B_1A_2B_2A_3B_3$	Path test
2	001010101	
6	101010101	Rising transition through path $C_0 \rightarrow C_4$
7	110101010	
3	010101010	Falling transition through path $C_0 \rightarrow C_4$
1	000000000	
4	011001100	
5	100110011	
8	111111111	

5.2

The following figure shows a two-bit shift register. Initially, both flip-flops are in the 0 state. The first two 0 inputs initialize the flip-flops to the 00 state. Subsequent inputs, outputs and state transitions are shown in the figure.

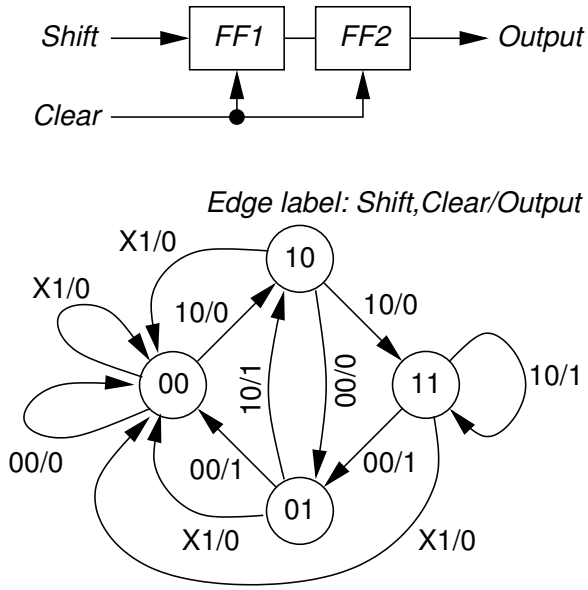




State diagram of 2–bit shift register.

5.3

The following figure shows a two-bit shift register with a *Clear* input and the state diagram.



State diagram of 2–bit shift register with clear input.

A necessary condition for an Eulerian path that will cover all edges traversing each edge exactly once is that the *indegree* must equal *outdegree* at each vertex. Since the state diagram does not satisfy this condition, an Eulerian path is not possible.

Notice that the above is only a necessary (not a sufficient) condition. Another condition, which is satisfied in this case, is that the graph should be *strongly connected.*

5.4

The longest path in the circuit (see Figure 5.2) is C_0 to C_4. The delay of this path should be tested for both rising and falling transitions. As shown in Example 5.3, the path delay for a rising transition is tested by vector 2 followed by 6, which causes the transition to ripple through the path. Similarly, the path delay for a falling transition can be tested by vector-pair, 6 followed by 2. From Table 5.2, vectors 1 through 6 cover all stuck-at faults. Since the circuit is combinational, these vectors can be applied in any order. We construct a sequence of seven vectors using these six vectors that contains the two delay test vector-pairs. The sequence is 1, 2, 6, 2, 3, 4, 5.

Note: If we use the result of Table 5.3, another sequence of six vectors, 6, 2, 6, 5, 4, 3, for all stuck-at faults and two path delay faults can be constructed.

5.5

In a combinational circuit, the fault activity is completely determined by the present vector irrespective of the previous vectors. Therefore, the faults detected by a vector remain the same irrespective of the position of the vector in the sequence. More importantly, the reverse order cannot reduce the overall fault coverage of any set of vectors. In a sequential circuit, the fault activity caused by a vector also depends on the circuit state caused by the previous vectors. The set of faults detected by a vector, therefore, varies depending on which vectors precede it. The total coverage of a sequence of vectors, applied to a sequential circuit in the reverse order, can be quite different from their original coverage. Although it can increase sometimes, mostly the overall coverage is found to decrease.

5.6

(a) Behavioral simulator: VHDL or Verilog circuit model, clock cycle accurate timing.
(b) Circuit-level simulator: e.g., Spice.
(c) Switch-level, or mixed-mode logic (with MOS capability), or circuit-level simulator.
(d) Multiple-delay logic, or circuit-level simulator.
(e) Unit-delay logic simulator.

5.7

When the two control inputs are changed to 0, the bus will be in the "floating" state and will retain its previous state, which is 1. Thus, the output of the inverter will remain 0.

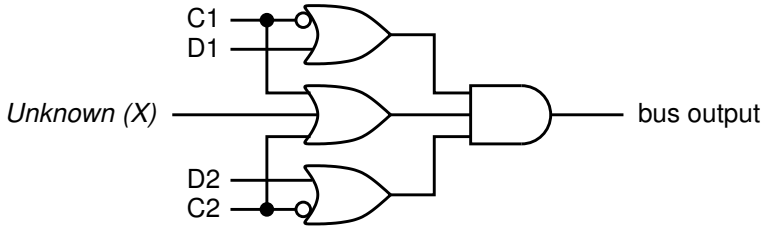
In the logic model of Figure 5.7, initially all four inputs to the bus driver (shaded block) may be 1. That will set the bus node to 1 and the output to 0 states. When the two control inputs (top and bottom inputs to AND gates in the shaded block) are changed to 0, the bus output will change to 0 and the output will change to 1. **Thus, the logic model gives incorrect values.**

5.8

The following circuit models an AND bus. This is a combinational model, which does not have memory. When both controls are off, $C1 = C2 = 0$, and *unknown* (X) value appears at the output. Logic simulators are often designed to supply the X value. The circuit can be modified to produce a 0 or a 1, instead of X. The bus will be set to a 0 if the X input of the OR gate is removed. It will produce a 1 if the three-input OR gate is omitted.

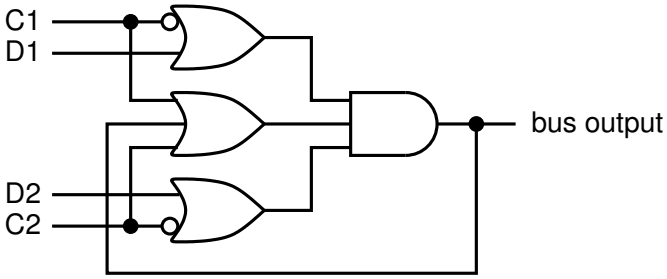
When only one control, $C1$ or $C2$, is turned on, the corresponding data, either $D1$ or $D2$, appears at the output. When both controls are on, the output is $D1 \cdot D2$.

Besides the lack of memory, this model does not also have the bidirectional behavior that is usually present in MOS circuits.



5.9

The following schematic shows a logic model for a bus with memory. When both drivers feed data to the bus, i.e., $C1 = C2 = 1$, $D1 \cdot D2$ appears at the output, assuming a 0-dominance. When both drivers are turned off, i.e., $C1 = C2 = 0$, the output retains its value through feedback. When only one driver is on, the corresponding data input appears at the output.



This model represents most of the characteristics of a MOS bus, with the exception of bidirectionality. One problem with it is that it is an asynchronous sequential circuit and cannot be correctly simulated by some simulators. An event-driven logic simulator can simulate it, but will be inefficient in comparison with synchronous circuit simulation.

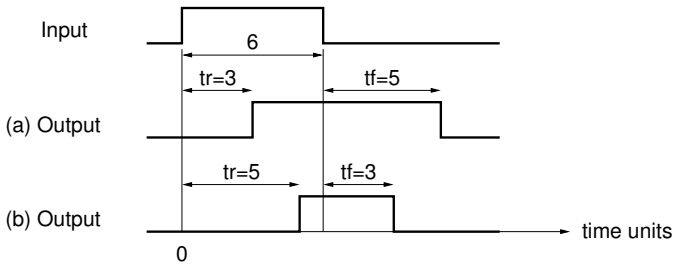
5.10

With the given inputs, 00, and output X, when the clock is applied the circuit will not be initialized. The reason is that in a three-state logic system the inversion of X is also X.

The circuit can be initialized to a 1 output by clocking the flip-flop when a 11 input is applied. Then, if we change the input to 10 and clock the flip-flop, the output will become 0. These two vectors can be correctly simulated by a three-state logic simulator.

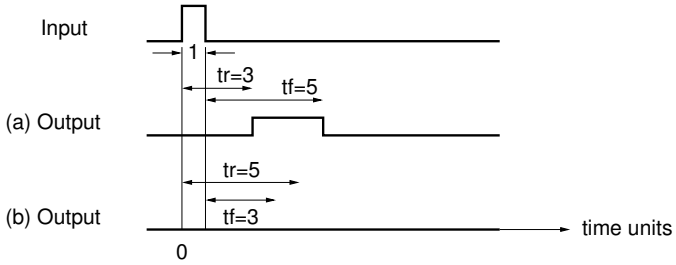
5.11

The two cases are sketched below. The rise and fall delays of the OR gate are denoted by tr and tf, respectively. In (a) the output pulse width is 8 units and in (b) it is 4 units.



5.12

The two cases are sketched below. The rise and fall delays of the OR gate are denoted by tr and tf, respectively. In (a), a rise is first scheduled to occur at 3 time units after the rising edge of the input. Before this rise takes place, the input falls at 1 unit, and reschedules a falling output at time 6 units. A conservative simulator produces an unknown (X) output between 3 and 6 units of time. This is shown as a level between logic 0 and 1 in the following figure.



In (b), the output cannot rise until 5 units of time. Meanwhile, at time unit 1, a fall is scheduled to be completed at time unit 4. Thus, the output does not change at all.

Case (b) is an example of a pulse being filtered by a slow gate. In simulators, this phenomenon is referred to as *spike suppression*. The actual waveform produced by the simulator depends upon the specific assumptions made. In pessimistic simulation, a pulse of ambiguous height may be produced as in case (a) above. In optimistic simulation, the output may remain unchanged if the input pulse width is smaller than the gate delay.

5.13

Upon the evaluation of a zero-delay gate, if the output changes then the new event is added to the current event list. Thus, all zero-delay events would be processed before the current event list becomes empty and the time is advanced.

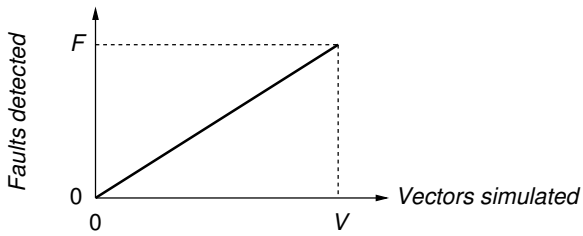
5.14

For unit-delay simulation only two time slots are needed: *current-time* and *next-time*. When the current-time event list becomes empty, the time pointer is moved

to the next-time slot, which then becomes the new current-time. The old current-time slot has no use now and it used as the next-time. Thus, the circular time-wheel contains only two slots.

5.15

Let us assume that the circuit has F faults and V vectors detect all faults. According to the given information, the coverage rises linearly from 0 to F faults as the number of simulated vectors increases from 0 to V. This is shown in the figure below. Assuming τ to be the CPU time required to simulate one vector in the true-value



mode, simulation time for the fault-free circuit is τV. When no fault-dropping is done, each faulty circuit is simulated for all V vectors. Therefore, the simulation time is given by,

$$T(no\ fault\ dropping) = \tau V + F \times \tau V = \tau V(1 + F)$$

Since faults are uniformly detected, each vector detects F/V new faults not detected by the previous vectors. Thus, the fault simulator divides the fault set into V equal subsets, each containing F/V faults. The F/V faulty circuits corresponding to the faults detected on the first vector are simulated through just one vector requiring a time $\tau F/V$. The F/V faulty circuits corresponding to the faults detected on the second vector are simulated through two vectors requiring a time $2\tau F/V$. Similarly, the F/V faulty circuits corresponding to the faults detected on the ith vector are simulated through i vectors requiring a time $i\tau F/V$. When we include the simulation time for the fault-free circuit, the total CPU time for simulation with fault dropping is given by,

$$T(fault\ dropping) = \tau V + \sum_{i=1}^{V} \frac{i\tau F}{V} = \tau V(1 + \frac{F}{2} \cdot \frac{1+V}{V})$$

For large F and V, we find

$$\frac{T(fault\ dropping)}{T(no\ fault\ dropping)} \approx 0.5$$

5.16

Since no fault dropping is used, the serial fault simulator must simulate the entire circuit $n + 1$ times. Assuming the CPU time for one simulation with all vectors is

t, total time of serial fault simulation is given by,

$$T(serial) = t(n+1)$$

Using CPU time t, the parallel simulator processes $w-1$ faults. Thus, it will make $n/(w-1)$ such passes, requiring total time,

$$T(parallel) = \frac{tn}{w-1}$$

Therefore,

$$\frac{T(serial)}{T(parallel)} = \frac{(n+1)(w-1)}{n}$$

5.17

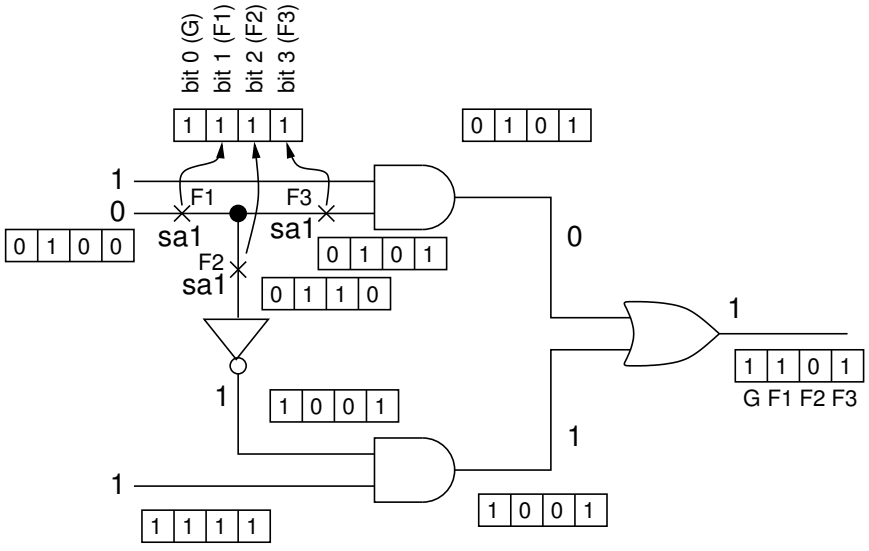
The circuit of Figure 5.22 is shown below. The bits of the four-bit word are assigned as follows:

Bit 0: G, good circuit

Bit 1: Faulty circuit with fault F1, second input s-a-1

Bit 2: Faulty circuit with fault F2, input to inverter s-a-1

Bit 3: Faulty circuit with fault F3, second input of first AND gate s-a-1



The figure shows the good and faulty circuit values for each signal by a four-bit word. A comparison among bits of the word at the primary output indicates that only the bit corresponding to F2 differs with the good circuit output. Hence, the vector 101 detects F2 but does not detect F1 and F3.

5.18

The following table shows how the fault effect (D or $\overline{D}$) propagates through an exclusive-OR gate $c = a \oplus b$. For a fault to affect the value of c, it should affect the value of a, or that of b, but not those of both. Therefore, for a fault to be included in L_c, it should be either in L_a or in L_b, but not in both. Thus,

$$\begin{aligned} L_c &= (L_a \cap \overline{L_b}) \cup (\overline{L_a} \cap L_b) \cup c_1, \text{ if } c = 0 \\ \text{or } L_c &= (L_a \cap \overline{L_b}) \cup (\overline{L_a} \cap L_b) \cup c_0, \text{ if } c = 1. \end{aligned}$$

One input	Other input	Output, c
D	D	0
$\overline{D}$	$\overline{D}$	0
D	$\overline{D}$	1
D	0	D
$\overline{D}$	0	$\overline{D}$
D	1	$\overline{D}$
$\overline{D}$	1	D

5.19

The two simulators differ in the dynamic memory usage. The major part of the memory used by each simulator consists of the lists that are stored for each line of the circuit. These lists are dynamic and continuously change as the simulation progresses. In a deductive fault simulator, the list for a line contains the faults that affect the value of that line. In a concurrent fault simulator, the list for a line contains all faults that affect the gate producing the signal on the line. Thus, the list may contain some faults that do not affect the output line but only affect the inputs of the gate. Since such faults will not be included in the fault list of that line in a deductive fault simulator, the corresponding list will be shorter.

Having a complete picture of faulty-circuit gates (i.e., its input and output signal values) allows the concurrent simulator to accurately simulate the events occurring at gates. In general, when gates have different rise and fall delays, the good-circuit events and various faulty-circuit events on a line can occur at different times. The timing in a deductive fault simulator basically follows the events of the good-circuit.

The other advantage that the expanded data-structure provides to the cuncurrent fault simulator is the ability to simulate a variety of non-Boolean and high-level gates.

5.20

(a) Though all four types of simulators can be used, deductive and parallel algorithms will experience significant slow down due to embedded memory blocks that are usually simulated at the functional level. Complexity of the parallel algorithm will also increase due to the non-Boolean signal states, X and Z. In

the deductive algorithm, the fault list propagation for these signal states will be complex and sometimes approximate.

(b) The best choice will be a concurrent fault simulator.

5.21

(a) When the tests can detect both single faults, there is high probability of detecting the multiple fault. Only in the rare case that the two faults mask each other (two-way masking) will the multiple fault go undetected.

(b) The test will not detect the multiple fault only if f2 masks the effect of f1 produced by the test. Usually, the probability of this one-way masking is small, but it is higher than the two-way masking in (a).

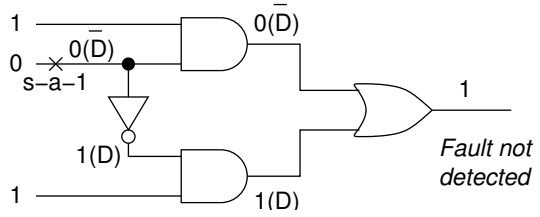
(c) In the majority of cases where single faults f1 and f2 are not detectable by the tests, the multiple fault (f1,f2) may also go undetected. However, in a special case where a test activates both single faults but fails to propagate the fault effect to a primary output, the two fault effects may cooperatively propagate to the output, detecting the fault.

5.22

For N faults, we effectively simulate the good circuit and N faulty copies of the circuit, each containing one fault. Suppose the circuit has G gates. At any time about half of the faulty circuit gates are identical to the good circuit gates because the corresponding faults are not active. Gates in these circuits are not explicitly simulated by the concurrent fault simulator. For the remaining $N/2$ circuits in which some signals differ from the good circuit, only a fraction α of gates actually differ from their counterparts in the good circuit. Taking all this into account, the concurrent fault simulator will only evaluate $G + \alpha GN/2$ gates, which is $1 + \alpha N/2$ times the gates simulated in a true-value simulation.

5.23

The following figure shows the TEST-DETECT procedure. First, true-value simulation determines the values for all lines. These are the binary values shown in the figure. Next, we start at the fault site. The value 0 activates the s-a-1 fault as $\overline{D}$.

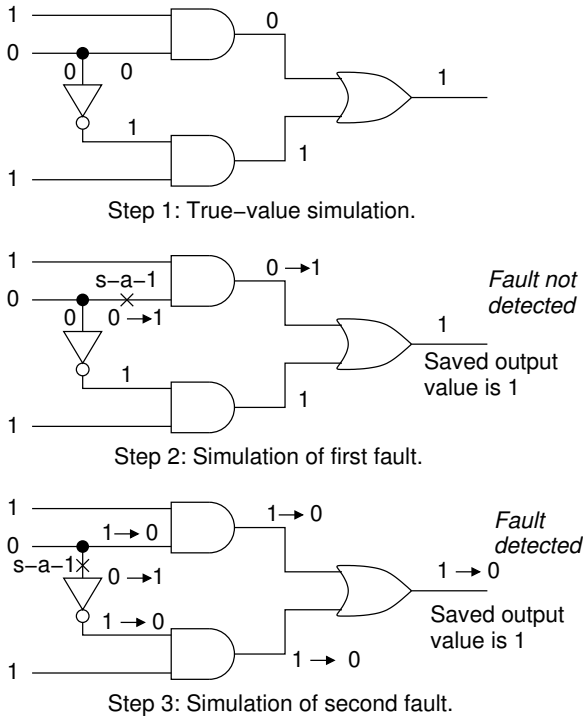


So, we temporarily replace 0 with a $\overline{D}$. This is shown as $0(\overline{D})$. Now $\overline{D}$ fans out to

the upper AND gate and the NOT gate. For the present signal states, the outputs of these gates are written as $0(\overline{D})$ and $1(D)$, respectively. The D at the output of the NOT gate propagates through the lower AND gate whose output is written as $1(D)$. Inputs $\overline{D}$ and D at the inputs of the OR gate leave the output unchanged as 1. **Thus, the given fault is not detected.**

5.24

Differential fault simulation of two faults requires three steps illustrated in the figure:



Step 1: True-value simulation. All line values are determined for the given input vector, 101, using logic simulation. The primary output value, 1 in this case, is saved.

Step 2: Simulation of first fault. Since the existing value at the site of the s-a-1 fault is 0, we place a $0 \rightarrow 1$ event there. Event-driven logic simulation propagates events until no more events exist. If the new output value differs from the saved true-value output, then the fault will be detected. In this case it is not detected.

Step 3: Simulation of second fault. We simultaneously restore the values to fault-free states and compute the values corresponding to the second fault. This is done by placing a $1 \rightarrow 0$ event at the site of the first fault and a $0 \rightarrow 1$ event at

the site of the second fault. Event-driven logic simulation now determines the output of the second faulty circuit, which is 0 in this case. Since this differs from the saved good circuit output value of 1, this fault is detected.

The following comments are relevant here:

- It is possible that the first fault changes the value at the site of the second fault. In that case, Step 3 will begin with just one event, because the second fault will be inactive for the signal states at the end of Step 2. However, as the simulation proceeds in Step 3, the second fault will become active and the second event will be placed.
- If there are no more faults to be simulated and another vector is to be simulated, then corresponding primary input changes are placed as events. True-value and fault simulation steps are successively repeated.

5.25 Fault sampling

Since the size of fault population ($N_p = 10^5$) is very large compared to the sample size ($N_s = 4,000$), we use the approximation of Equation 5.5 (page 123 in the book.)

$$\text{Sample coverage, } x = \frac{3,900}{4,000} = 0.975$$

Using Equation 5.8 (see page 123 in the book), we get

$$\begin{aligned} 3\sigma \text{ coverage estimate} &= x \pm \frac{4.5}{N_s}\sqrt{1 + 0.44 N_s x(1-x)} \\ &= 0.975 \pm \frac{4.5}{4,000}\sqrt{1 + 0.44 \times 4,000 \times 0.975 \times 0.025} \\ &= 0.975 \pm 0.0075 \text{ or } 97.50 \pm 0.75 \text{ percent} \end{aligned}$$

5.26 Fault sampling

Assuming that the fault sample size is much smaller than the total fault population, i.e., $N_s \ll N_p$, we use the result of Equation 5.9 (page 124 in the book), which can be written as,

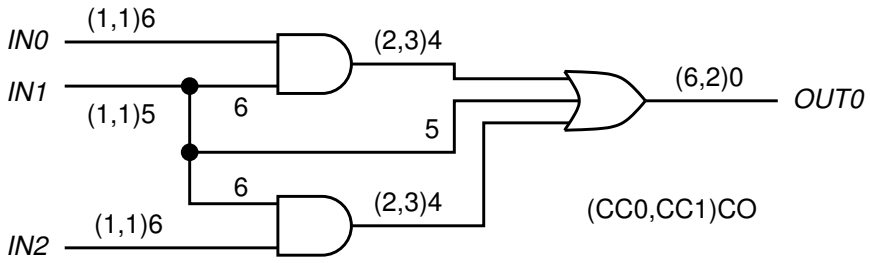
$$\text{Sample size, } N_s = \frac{4.5^2}{\Delta^2} 0.44 x(1-x)$$

where $\pm\Delta$ is the 3σ range of the coverage estimate and x is the sample coverage. Using the given data, $\Delta = 0.02$ and $x = 0.70$, we obtain

$$N_s = \frac{4.5^2 \times 0.44 \times 0.7 \times 0.3}{0.02^2} = 4,678 \text{ faults}$$

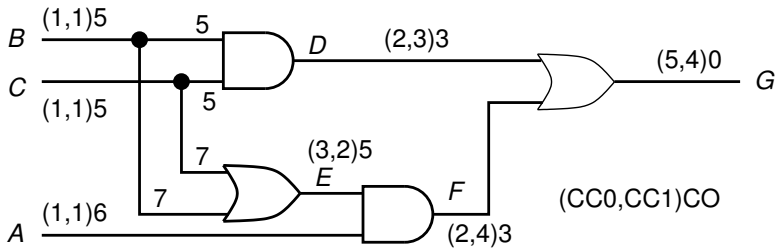
Chapter 6: Testability Measures

6.1 SCOAP



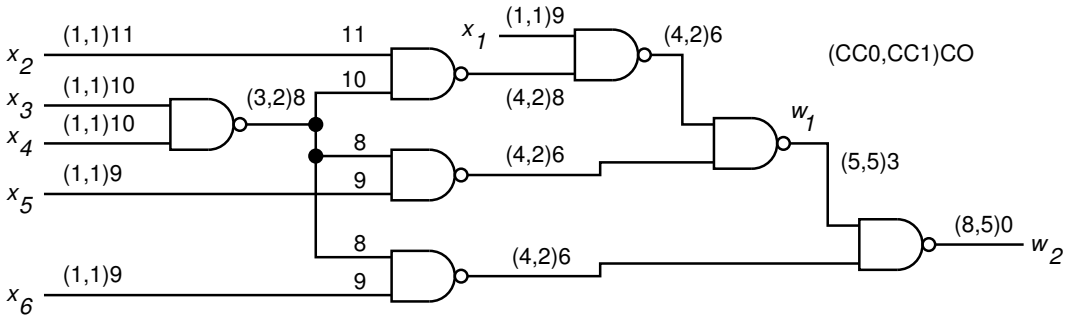
Circuit of Figure 6.1 with combinational SCOAP measures.

6.2 SCOAP



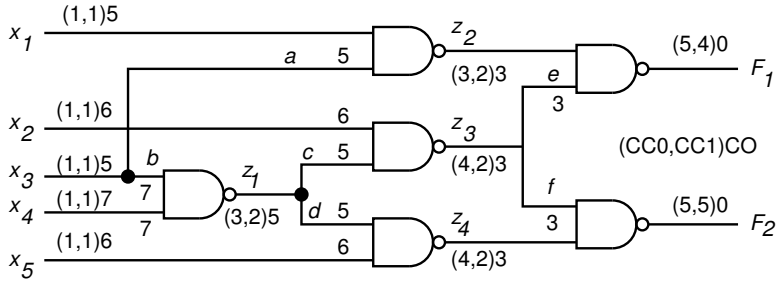
Circuit of Figure 6.20 with combinational SCOAP measures.

6.3 SCOAP



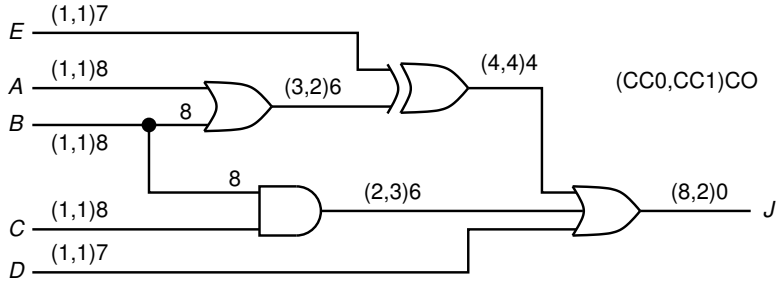
Circuit of Figure 6.21 with combinational SCOAP measures.

6.4 SCOAP



Circuit of Figure 6.22 with combinational SCOAP measures.

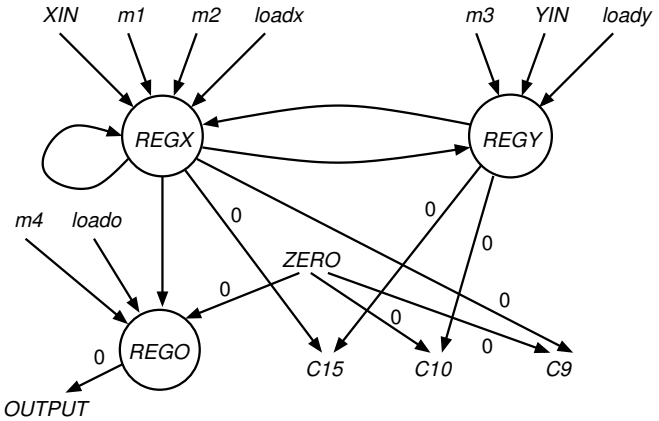
6.5 SCOAP



Circuit of Figure 6.23 with combinational SCOAP measures.

6.6 High-level testability

The data flow graph (DFG) of the given circuit shown below. The table gives the sequential depth testability measures for all input-output pairs of signals.

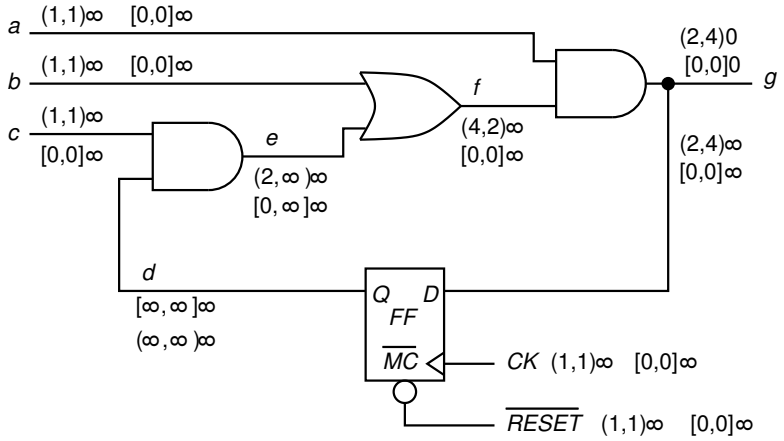


Data flow graph (DFG) for the circuit of Figure 6.24.

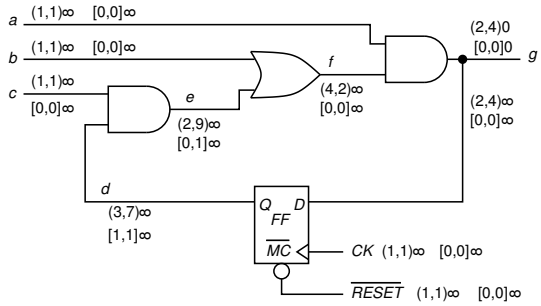
Input signal	Output signal						
	$OUTPUT$	$C9$	$C15$	$C10$	$REGX$	$REGY$	$REGO$
XIN	2	1	1	2	1	2	2
$m1$	2	1	1	2	1	2	2
$m2$	2	1	1	2	1	2	2
$loadx$	2	1	1	2	1	2	2
$m3$	3	2	1	1	2	1	3
YIN	3	2	1	1	2	1	3
$loady$	3	2	1	1	2	1	3
$m4$	1						1
$loado$	1						1
$ZERO$	1	0		0			1
$REGX$	1	0	0	1	1	1	1
$REGO$	0						
$REGY$	2	1	0	0	1	2	2

6.7 SCOAP

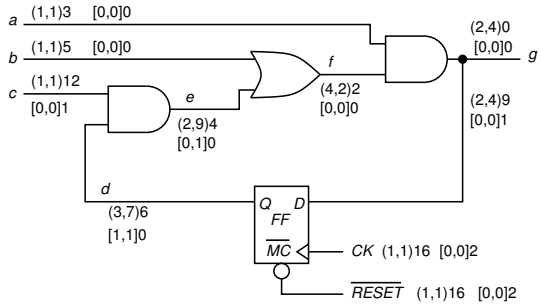
The steps of calculation for SCOAP testability measures are shown in the three figures that follow. Combinational measures are shown as $(CC0, CC1)CO$ and sequential measures as $[SC0, SC1]SO$.



Circuit of Figure 6.25: PI and PO initialization and first controllability pass.



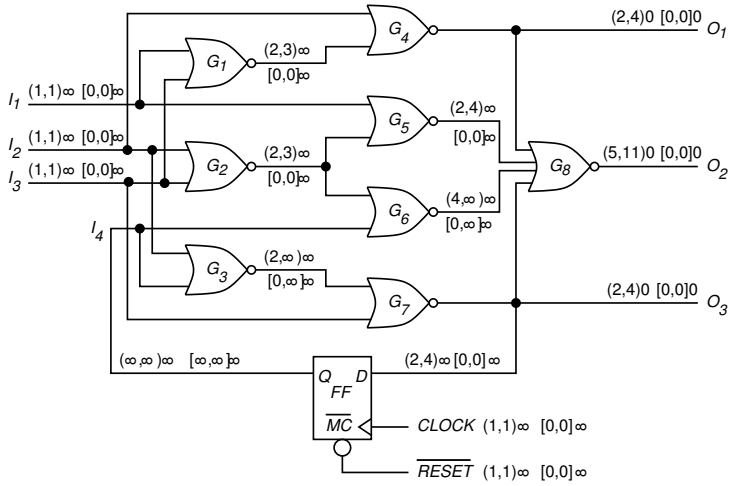
Circuit of Figure 6.25: Converged controllability values.



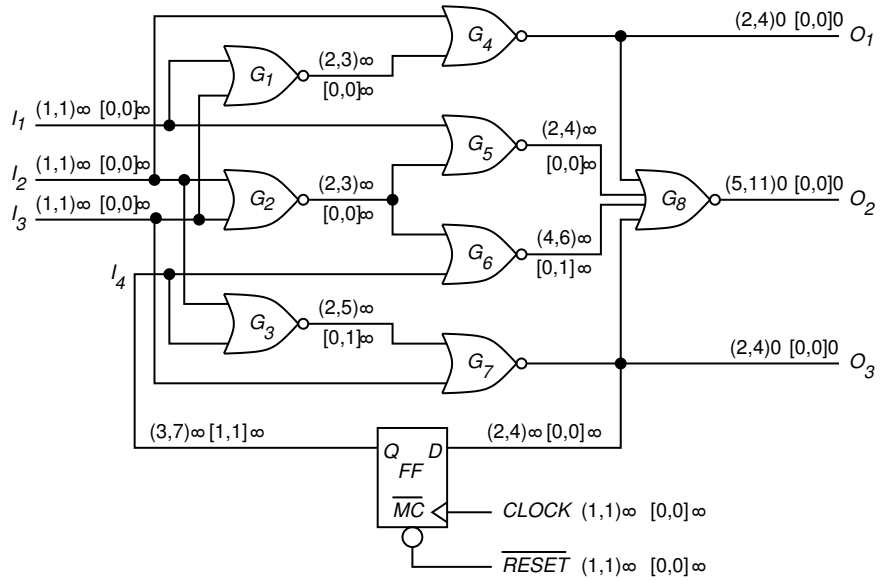
Circuit of Figure 6.25: All controllability and observability values.

6.8 SCOAP

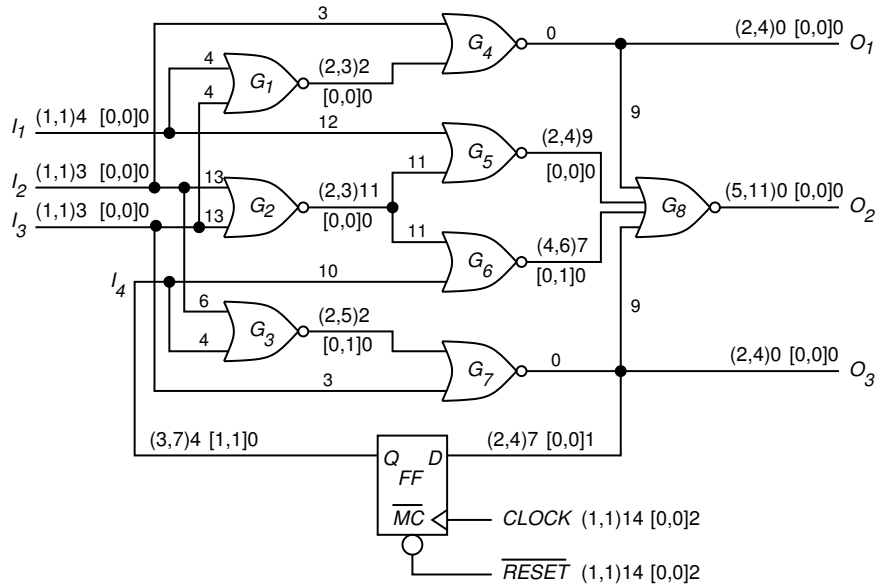
The steps of calculation for SCOAP testability measures are shown in the three figures that follow. Combinational measures are shown as $(CC0, CC1)CO$ and sequential measures as $[SC0, SC1]SO$.



Circuit of Figure 6.26: PI and PO initialization and first controllability pass.



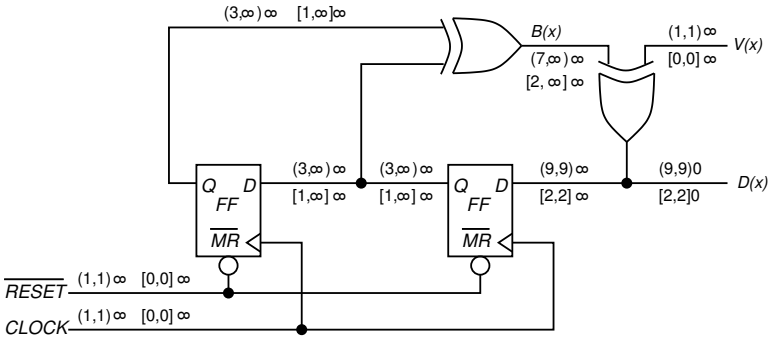
Circuit of Figure 6.26: Converged controllability values.



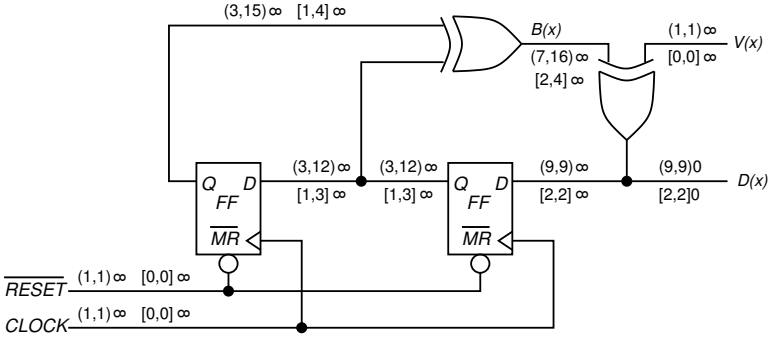
Circuit of Figure 6.26: All controllability and observability values.

6.9 SCOAP

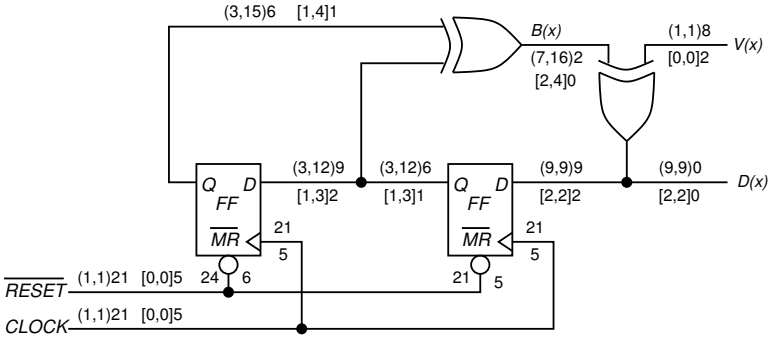
The steps of calculation for SCOAP testability measures are shown in the three figures that follow. Combinational measures are shown as $(CC0, CC1)CO$ and sequential measures as $[SC0, SC1]SO$.



Circuit of Figure 6.27: Initialization and first controllability pass.



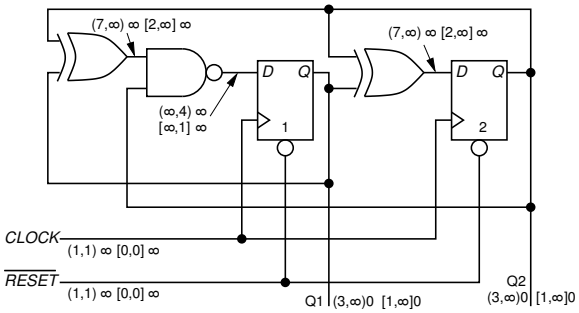
Circuit of Figure 6.27: Converged controllability values.



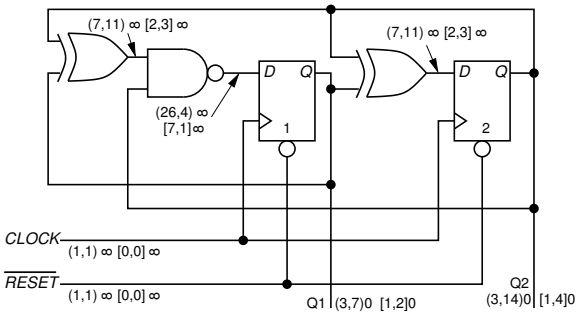
Circuit of Figure 6.27: All controllability and observability values.

6.10 SCOAP

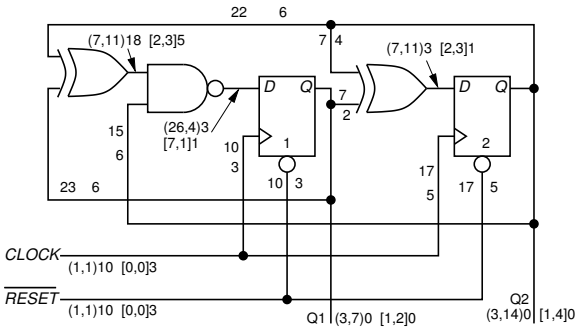
The steps of calculation for SCOAP testability measures are shown in the three figures that follow. Combinational measures are shown as $(CC0, CC1)CO$ and sequential measures as $[SC0, SC1]SO$.



Circuit of Figure 6.28: Initialization and first controllability pass.



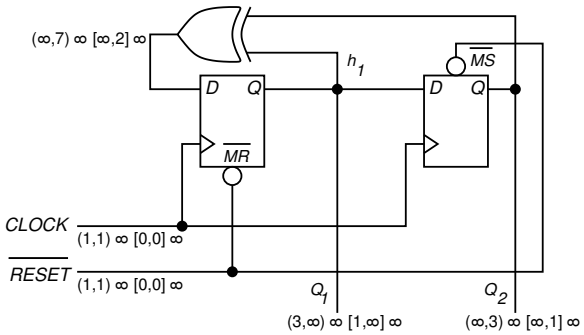
Circuit of Figure 6.28: Converged controllability values.



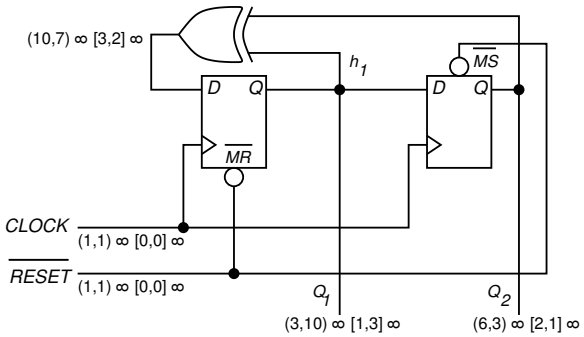
Circuit of Figure 6.28: All controllability and observability values.

6.11 SCOAP

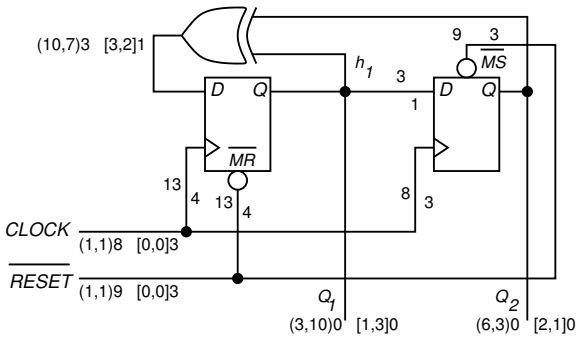
The steps of calculation for SCOAP testability measures are shown in the three figures that follow. Combinational measures are shown as $(CC0, CC1)CO$ and sequential measures as $[SC0, SC1]SO$.



Circuit of Figure 6.29: Initialization and first controllability pass.



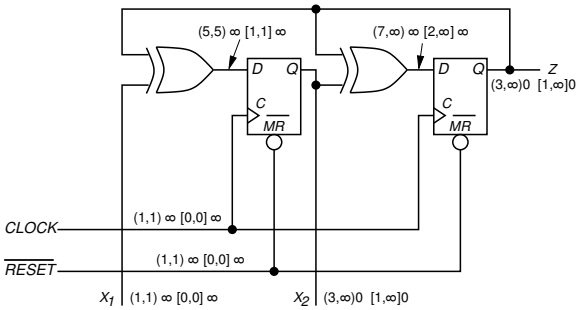
Circuit of Figure 6.29: Converged controllability values.



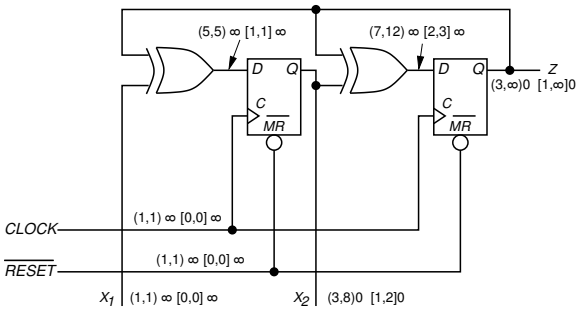
Circuit of Figure 6.29: All controllability and observability values.

6.12 SCOAP

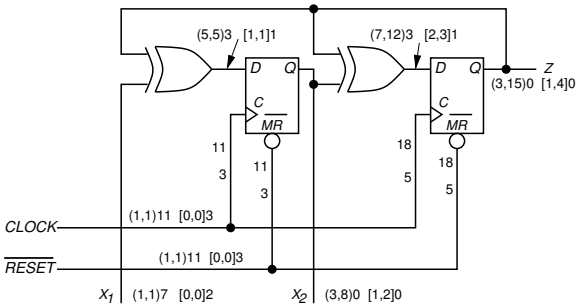
The steps of calculation for SCOAP testability measures are shown in the three figures that follow. Combinational measures are shown as $(CC0, CC1)CO$ and sequential measures as $[SC0, SC1]SO$.



Circuit of Figure 6.30: Initialization and first controllability pass.



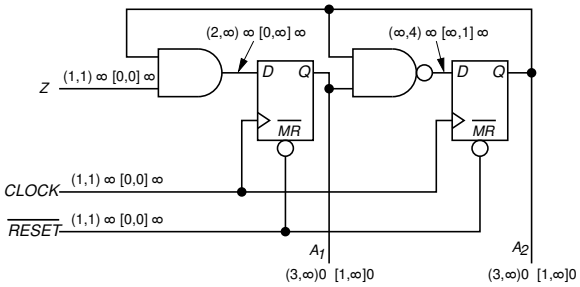
Circuit of Figure 6.30: Continuation of controllability calculations.



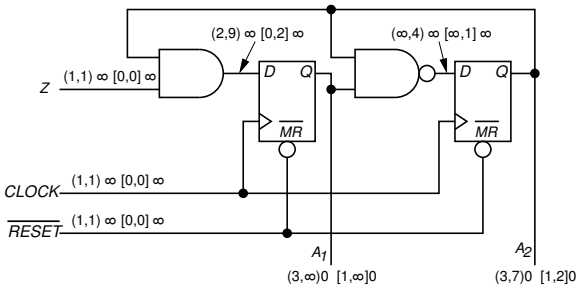
Circuit of Figure 6.30: Stabilized controllability and observability values.

6.13 SCOAP

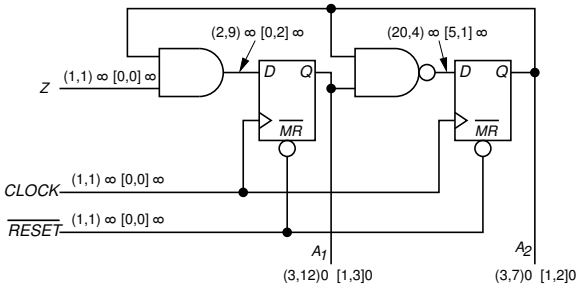
The steps of calculation for SCOAP testability measures are shown in the four figures that follow. Combinational measures are shown as $(CC0, CC1)CO$ and sequential measures as $[SC0, SC1]SO$.



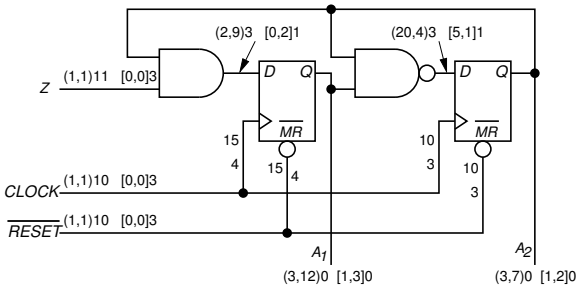
Circuit of Figure 6.31: Initialization and first controllability pass.



Circuit of Figure 6.31: Continuation of controllability calculation.



Circuit of Figure 6.31: Stabilized controllability values.

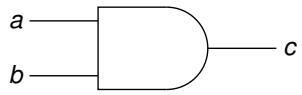


Circuit of Figure 6.31: All controllability and observability values.

Chapter 7: Combinational Circuit ATPG

7.1 Cubes

AND gate:



Singular cover:

a	b	c
0	X	0
X	0	0
1	1	1

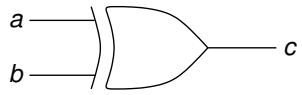
Propagation D cubes (last two cubes are not propagation D-cubes since they do not propagate D or $\overline{D}$):

a	b	c
1	D	D
D	1	D
D	D	D
1	$\overline{D}$	$\overline{D}$
$\overline{D}$	1	$\overline{D}$
D	$\overline{D}$	0
$\overline{D}$	D	0

Primitive D cube of failure for a sa1:

a	b	c
0	1	$\overline{D}$

Exclusive-OR gate:



Singular cover:

a	b	c
0	1	1
1	0	1
1	1	0
0	0	0

Propagation D cubes (last four cubes are not propagation D-cubes since they do not papagate D or $\overline{D}$):

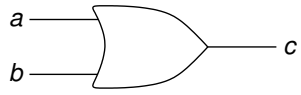
a	b	c
0	D	D
D	0	D
D	1	$\overline{D}$
1	D	$\overline{D}$
0	$\overline{D}$	$\overline{D}$
$\overline{D}$	0	$\overline{D}$
1	$\overline{D}$	D
$\overline{D}$	1	D
D	D	0
D	$\overline{D}$	1
$\overline{D}$	D	1
$\overline{D}$	$\overline{D}$	0

Primitive D cubes of failure for a sa1:

a	b	c
0	0	$\overline{D}$
0	1	D

7.2 Stuck-at fault testing

(a) *Three tests for a two-input OR gate:*



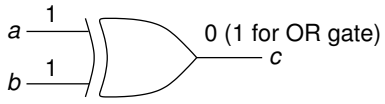
Vector number	a	b	c	Collapsed faults tested
1	0	0	$\overline{D}$	a sa1, b sa1, c sa1
2	0	1	D	b sa0, c sa0
3	1	0	D	a sa0, c sa0

(b) *Gate replacements:*

OR replaced by:	Test results		
	Vector 1	Vector 2	Vector 3
AND	pass	fail	fail
NAND	fail	pass	pass
NOR	fail	fail	fail

The three-vector test will detect the error if the OR gate were to be replaced by an AND, NAND or NOR gate.

(c) *OR gate replaced by an exclusive-OR gate:* All three vectors will produce the same output as that of the OR gate. Therefore, this error will not be detected. It is necessary to include a fourth vector 11 to detect this error. The addition of the



fourth vector makes the vector set exhaustive, which completely verifies the truth table of the gate.

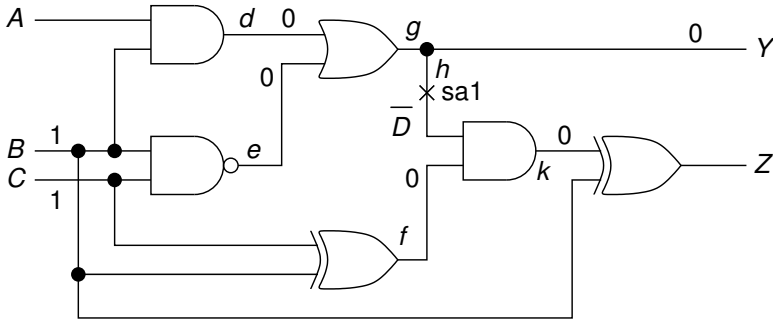
Note: In a simulation-based comparison of two circuits to establish logic equivalence, a good (*though not complete*) heuristic is to use a vector set that covers all single stuck-at faults in both circuits. See the paper: V. D. Agrawal, "Choice of Tests for Logic Verification and Equivalence Checking and the Use of Fault Simulation," *Proc. 13th Int. Conf. VLSI Design*, 2000, pp. 306-311.

7.3 D-ALG

We level order the signals and proceed as follows:

Step no.	Action	Signals $A\ B\ C\ d\ e\ f\ g\ Y\ h\ k\ Z$	D front.	Impl. stack
1	Fault Activation	$0\ 0\ \overline{D}$	k	$g=0$
	Immediate impl.	$0\ 0\quad 0\ 0\ \overline{D}$	k	$g=0$
	Immediate impl.	$1\ 1\ 0\ 0\quad 0\ 0\ \overline{D}$	k	$g=0$
	Immediate impl.	$1\ 1\ 0\ 0\ 0\ 0\ 0\ \overline{D}$	k	$g=0$
	Immediate impl.	$1\ 1\ 0\ 0\ 0\ 0\ 0\ \overline{D}\ 0$	ϕ	$g=0$
	Immediate impl.	$1\ 1\ 0\ 0\ 0\ 0\ 0\ \overline{D}\ 0\ 1$	ϕ	$g=0$

The fault is redundant, because the D-frontier disappeared. No backtracks. Signals are shown in the following figure.

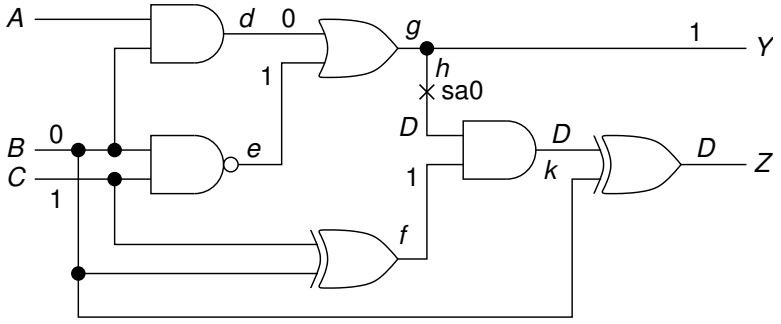


7.4 D-ALG

We level order the signals and proceed as follows:

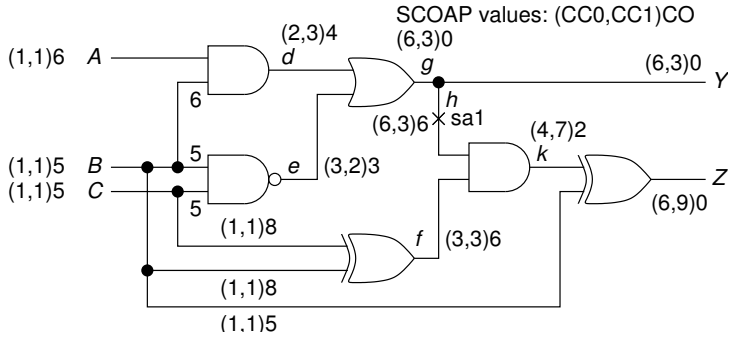
Step no.	Action	Signals $A\ B\ C\ d\ e\ f\ g\ Y\ h\ k\ Z$	D front.	Impl. stack
1	Fault activation	$1\ 1\ D$	k	$g=1$
2	D-drive $h \to k$	$1\ 1\ 1\ D\ D$	Z	$f=1$ $g=1$
3	D-drive $k \to Z$	$0\quad 1\ 1\ 1\ D\ D\ D$	PO	$B=0$ $f=1$ $g=1$
	Immediate Impl.	$0\quad 0\quad 1\ 1\ 1\ D\ D\ D$	PO	”
	Immediate Impl.	$0\quad 0\ 1\ 1\ 1\ 1\ D\ D\ D$	PO	”
	Immediate impl.	$0\ 1\ 0\ 1\ 1\ 1\ 1\ D\ D\ D$	PO	”

The test is: $A = X$, $B = 0$, $C = 1$ as shown in the following figure; 0 backtracks.



7.5 PODEM

The figure below shows the SCOAP testability measures used for guiding PODEM.



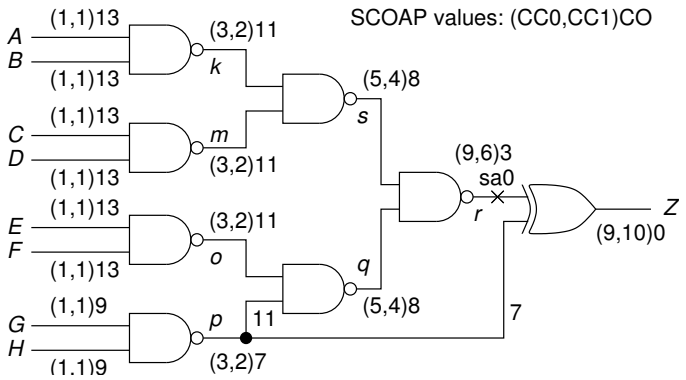
The steps of the PODEM algorithm are recorded in the following table:

Step No.	Objective	Action	Imp. stack	Implied signal values $A\ B\ C\ d\ e\ f\ g\ h\ k\ Y\ Z$	D front.	X path
1	$g=0$	Backtrace	$B=1$	1	ϕ	ok
2	$g=0$	Backtrace	$C=1$ $B=1$	1 1 0 0 0 1	ϕ	none
3	$g=0$	Backtrack	$C=0$ $B=1$	1 0 1 1 1 1 1 1 0	ϕ	none
4	$g=0$	Backtrack	$B=0$	0 0 1 1 1 1	ϕ	none
5	$g=0$	Backtrack	Empty			
Algorithm termination: Objective g=0 is impossible; fault h s-a-1 is redundant.						

Explanation: An *X-path* is a path from the fault site to a PO, such that the signals on it are either faulty states (D or $\overline{D}$) or *undetermined*. An "ok" for *X-path* in the table means that one or more such paths exist. Having no X-path is a reason for *backup* because its existence is a necessary condition for the detection of the fault. When a series of backups leads to an *empty* stack, it indicates that the objective $g=0$ is impossible. As a result, **the fault h s-a-1 cannot be activated and, hence, it is redundant.** Three backtracks.

7.6 PODEM

The figure below shows the SCOAP testability measures used for guiding PODEM.



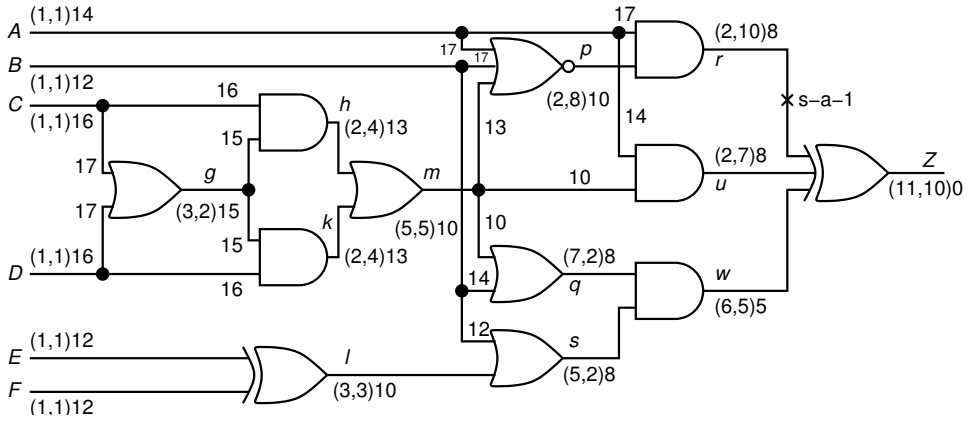
The steps of the PODEM algorithms are recorded in the following table:

Step No.	Objective	Action	Imp. stack	Implied signal values $ABCDEFGHkmopqsrZ$	D front.	X path
1	$r = 1$	Backtrace	$E = 0$	$E = 0, o = 1$	ϕ	ok
2	$r = 1$	Backtrace	$G = 0$ $E = 0$	$E = 0, G = 0, o = 1, p = 1$ $q = 0, r = 1, Z = \overline{D}$	PO	ok
Algorithm termination: Fault detected with 0 backtracks. *Test is* $\{ABCDEFGH\} = \{XXXX0X0X\}$						

Explanation: See the explanation in Problem 7.5.

7.7 PODEM and FAN

The following figure shows the SCOAP testability measures used for guiding PODEM and FAN.



Step No.	Objective	Action	Imp. stack	Implied signal values $ABCDEFghklmpqsruwZ$	D front.	X path
1	$r = 0$	Backtrace	$A = 0$	$r = 0(\overline{D}), u = 0$	Z	ok
2	$w = 1$ (D-drive)	Backtrace	$B = 1$ $A = 0$	$A = 0, B = 1, p = 0, q = 1, s = 1$ $r = 0(\overline{D}), u = 0, w = 1, Z = D$	PO	ok
Algorithm termination: Fault detected with 0 backtracks. Test is $\{ABCDEF\} = \{01XXXX\}$						

The following table gives the steps that PODEM takes:
For an explanation of *X-path*, see Problem 7.5.

(a) FAN ATPG. Step 1 is the same as for PODEM.

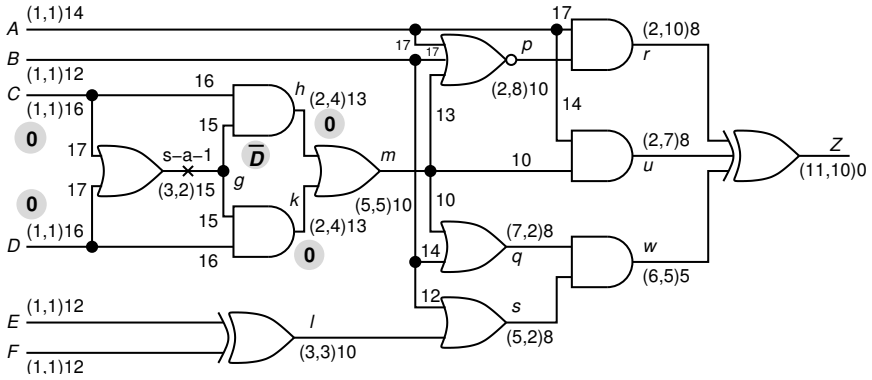
Step 2. Goal: propagate $\overline{D}$ from r to Z.
Goal: set $w = 1$, 1 vote for 1, set $q = 1$, 1 vote for 1, set $s = 1$, 1 vote for 1, set $B = 1$, 2 votes for 1, no votes for 0. Rest of step 2 is exactly the same as PODEM.
Headlines are m and l.
Initial objective: set $r = 0$.
Final objective: set $B = 1$.
Fanout objectives: set $B = 1$.
Head objectives: not used. 0 backtracks.

(b) The following lists dominators for all signals.

Gate	Dominators	Gate	Dominators	Gate	Dominators
Z	–	p	$r,\ Z$	h	$m,\ Z$
r	Z	q	$w,\ Z$	k	$m,\ Z$
u	Z	s	$w,\ Z$	g	$m,\ Z$
w	Z	m	Z	l	$s,\ w,\ Z$

7.8 PODEM

The figure below shows the SCOAP testability measures used for guiding PODEM.

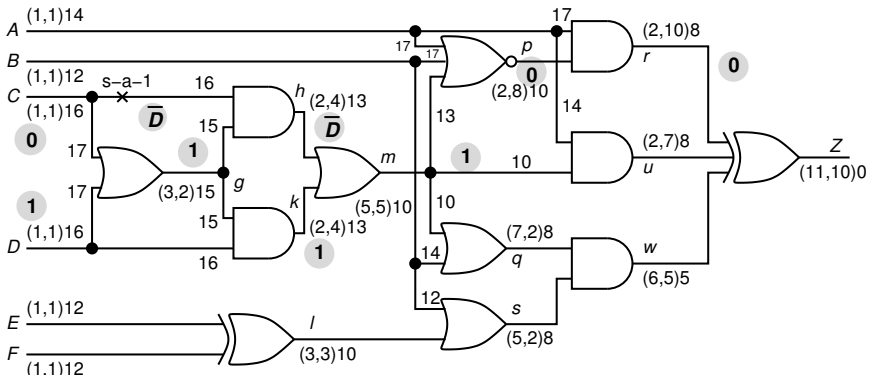


The following table gives the steps of PODEM (see Problem 7.5 for an explanation of *X-path*):

Step No.	Objective	Action	Imp. stack	Implied signal values $ABCDEFghklmpqsruwZ$	D front.	X path
1	$g = 0(\overline{D})$	Backtrace	$C = 0$	$C = 0, h = 0$	ϕ	ok
2	$g = 0(\overline{D})$	Backtrace	$D = 0$ $C = 0$	$C = 0, D = 0, g = 0(\overline{D})$ $h = 0, k = 0, m = 0, u = 0$	ϕ	none
3	$g = 0(\overline{D})$	Backtrack	$D = 1$ $C = 0$	$C = 0, D = 1, g = 1, h = 0$ $k = 1, m = 1, p = 0, q = 1, r = 0$	ϕ	none
4	$g = 0(\overline{D})$	Backtrack	$C = 1$	$C = 1, g = 1, h = 1, m = 1$ $p = 0, q = 1, r = 0$	ϕ	none
5	$g = 0(\overline{D})$	Backtrack	Empty			
Algorithm termination: $g = 0(\overline{D})$ with X-path impossible; fault g s-a-1 is redundant. 3 backtracks.						

7.9 PODEM

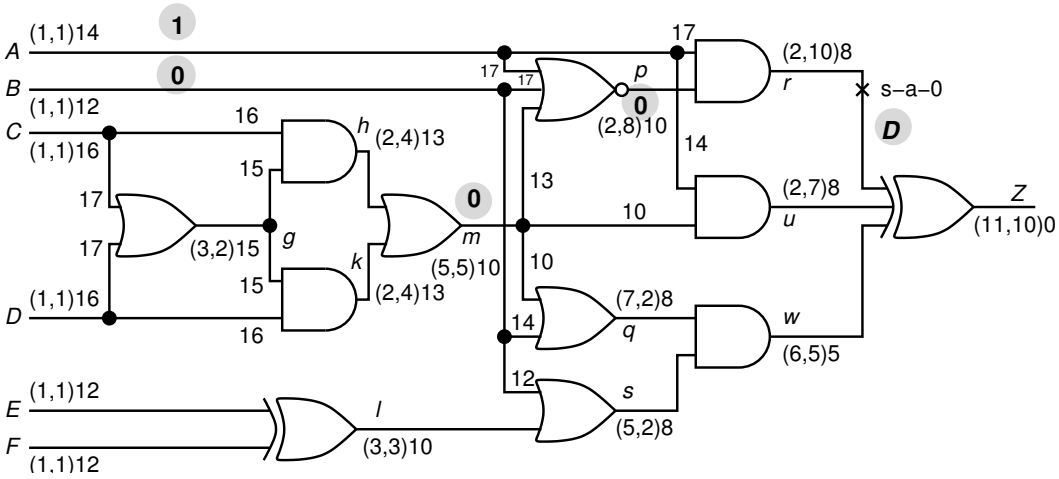
The following figure and table show the SCOAP testability measures and the steps of PODEM. See Problem 7.5 for an explanation of *X-path*.



Step No.	Objective	Action	Imp. stack	Implied signal values $ABCDEFghklmpqsruwZ$	D front.	X path
1	$C - h = 0(\overline{D})$	Backtrace	$C = 0$	$C = 0, C - h = \overline{D}$	h	ok
2	$h = 0(\overline{D})$	Backtrace	$D = 1$ $C = 0$	$C = 0, D = 1, C - h = \overline{D}, g = 1, k = 1$ $h = \overline{D}, m = 1, p = 0, q = 1, r = 0$	ϕ	none
3	$h = 0(\overline{D})$	Backtrack	$D = 0$ $C = 0$	$C = 0, D = 0, C - h = \overline{D}, g = 0, h = 0$ $k = 0, m = 0, u = 0$	ϕ	none
4	$C - h = 0(\overline{D})$	Backtrack	$C = 1$	$C = 1, g = 1, h = 1, m = 1, p = 0$ $q = 1, s = 1, r = 0, w = 1$	ϕ	none
5	$C - h = 0(\overline{D})$	Backtrack	Empty			
Algorithm termination: $C - h = 0(\overline{D})$ with X-path impossible; fault $C - h$ s-a-1 is redundant. 3 backtracks.						

7.10 FAN

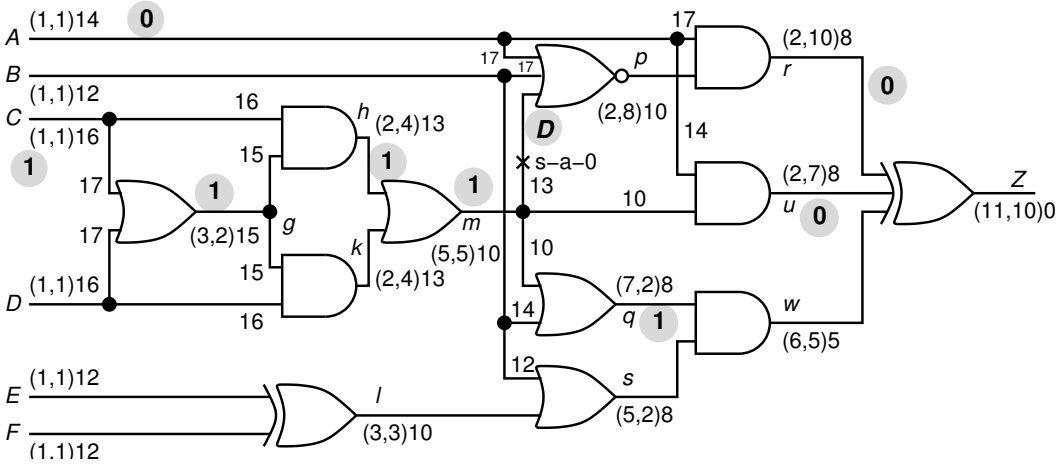
The following figure showsthe SCOAP testability measures used for FAN. The headlines are m and l.



Step 1: Goal – set $r = 1 \rightarrow$ set $A = p = 1 \rightarrow$ set $A = m = B = 0$
D-frontier $= \phi$.
Conflict at stem A, choose $A = 1$
Forward imply $A = 1, B = 0, p = 0, r = 0$, fault r s-a-0 cannot be sensitized–is redundant (0 backtracks).

7.11 SOCRATES

The following figure shows the SCOAP testability measures used for SOCRATES.



Static learning

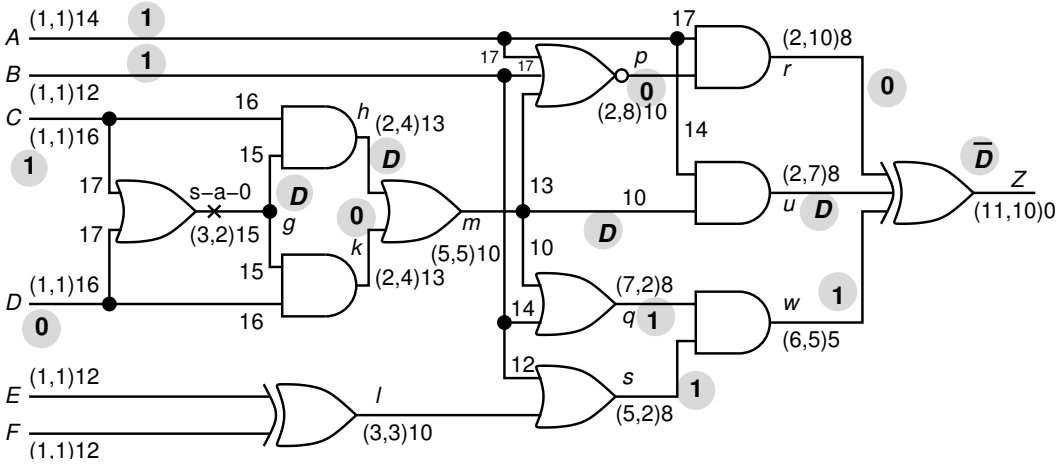
Signal	Learned implication
$B = 1$	$(w = 0) \Rightarrow (B = 0)$
$g = 0$	$(m = 1) \Rightarrow (g = 1)$
$C = 1$	$(h = 0) \Rightarrow (C = 0)$
$D = 1$	$(k = 0) \Rightarrow (D = 0)$

Step 1: Objective – set $m = 1$
Implication stack – $C = 1$
Assignments – $C = 1, g = 1, h = 1, m = 1, q = 1$
D-frontier – p
Dynamic learning – $(k = 0) \Rightarrow (D = 0)$, $(w = 0) \Rightarrow (l = 0)$

Step 2: Objective – set $A = 0$
Implication stack – $A = 0, C = 1$
Assignments – $C = 1, g = 1, h = 1, m = 1, q = 1, A = 0, r = 0, u = 0$
X-path check – fault propagation path blocked – alternative assignment $A = 1$ infeasible
Fault $m - p$ s-a-0 is redundant (no backtracks).
No applications of Modus Tollens or constructive dilemma.

7.12 SOCRATES

The following figure shows the SCOAP testability measures used for SOCRATES.



Static learning

Signal	Learned implication
$B = 1$	$(w = 0) \Rightarrow (B = 0)$
$g = 0$	$(m = 1) \Rightarrow (g = 1)$
$C = 1$	$(h = 0) \Rightarrow (C = 0)$
$D = 1$	$(k = 0) \Rightarrow (D = 0)$

Step 1: Objective – set $g = 1$
Implication stack – $C = 1$
Assignments – $C = 1, g = D, h = D$
D-frontier – k, m
Dynamic learning – $(D = 0) \Rightarrow (m = D)$

Step 2: Objective – set $D = 0$
Implication stack – $D = 0, C = 1$
Assignments – $C = 1, D = 0, g = D, h = D, k = 0, m = D$
D-frontier – p, q, u

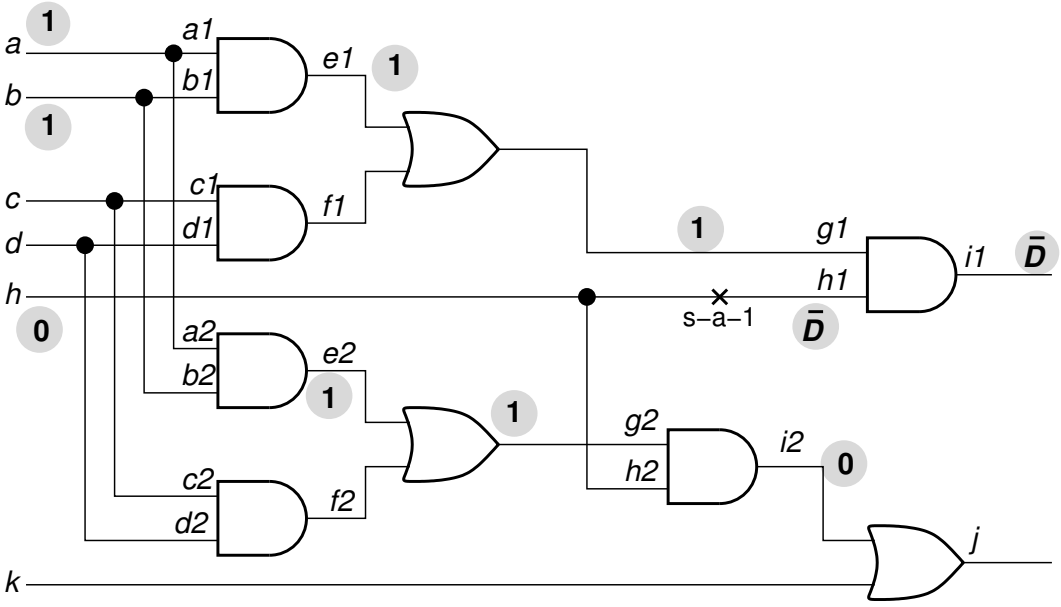
Step 3: Objective – Propagate D to u, set $A = 1$
Implication stack – $A = 1, D = 0, C = 1$
Assignments – $C = 1, D = 0, A = 1, g = D, h = D, k = 0, m = D, p = 0, r = 0, u = D$
D-frontier – Z
Dynamic learning – $(B = 0) \Rightarrow (q = D)$, $(B = 1) \Rightarrow (w = 1) \Rightarrow (Z = \overline{D})$

Step 4: Objective – Propagate D to Z, set $w = 1$
Implication stack – $B = 1, A = 1, D = 0, C = 1$
Assignments – $C = 1, D = 0, A = 1, B = 1, g = D, h = D, k = 0, m = D, p = 0, r = 0, u = D, q = 1, s = 1, w = 1, Z = \overline{D}$
Use learned implication $(w = 0) \Rightarrow (B = 0)$
Fault tested, no backtracks.
No applications of Modus Tollens or constructive dilemma.

7.13 D-ALG

Step	Action	Impl. stack	Forward implications	D-frontier
1	Fault act.	$h = 0$	$h = 0, h1 = \overline{D}, i2 = 0$	$i1$
2	D-prop.	$g1 = 1, h = 0$	$g1 = 1, h = 0, h1 = \overline{D}$ $i1 = \overline{D}, i2 = 0$	PO
3	Justify	$e1 = 1, g1 = 1$ $h = 0$	$e1 = 1, g1 = 1, h = 0$ $h1 = \overline{D}, i1 = \overline{D}, i2 = 0$	PO
4	Justify	$a = 1, b = 1$ $e1 = 1, g1 = 1$ $h = 0$	$a = 1, b = 1, e1 = 1, g1 = 1$ $e2 = 1, g1 = 1, g2 = 1$ $h = 0, h1 = \overline{D}, i1 = \overline{D}$ $i2 = 0$	PO
Test found: $(a, b, c, d, h, k) = (1, 1, X, X, 0, X)$; $i1 = \overline{D}$				

The following figure shows the circuit and the signal values specified by D-algorithm.

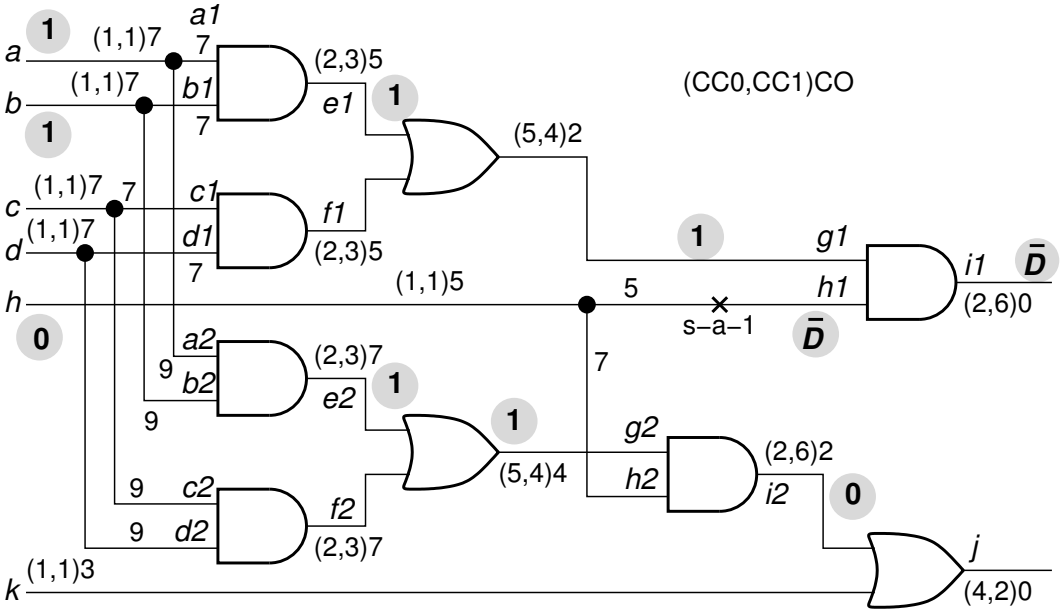


This test is found without any backtracks.

7.14 PODEM

Step	Objective (goal)	Impl. stack	Forward implications	D frontier	X path
1	Fault act.	$h = 0$	$h = 0, h1 = \overline{D}, i2 = 0$	$i1$	ok
2	Fault prop. $g1 = 1$	$h = 0, a = 1$	$a = 1, h = 0, h1 = \overline{D}$ $i2 = 0$	$i1$	ok
3	Fault prop. $g1 = 1$	$h = 0, a = 1$ $b = 1$	$a = 1, b = 1, e1 = 1$ $e2 = 1, g1 = 1, g2 = 1$ $h = 0, h1 = \overline{D}, i1 = \overline{D}$ $i2 = 0$	PO	ok
Test found: $(a, b, c, d, h, k) = (1, 1, X, X, 0, X)$; $i1 = \overline{D}$					

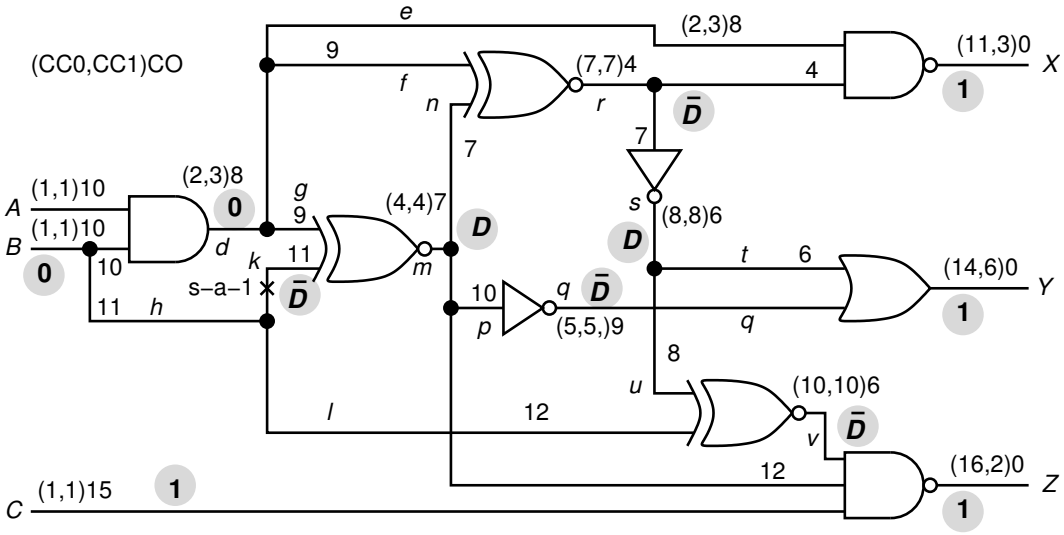
The following figure shows the SCOAP testability measures used to guide the PODEM algorithm, and the signal values detremined.



This test is found without any backtracks.

7.15 FAN

The following figure shows the SCOAP testability measures used to guide the ATPG. The signal velues are those determined by the steps described below.



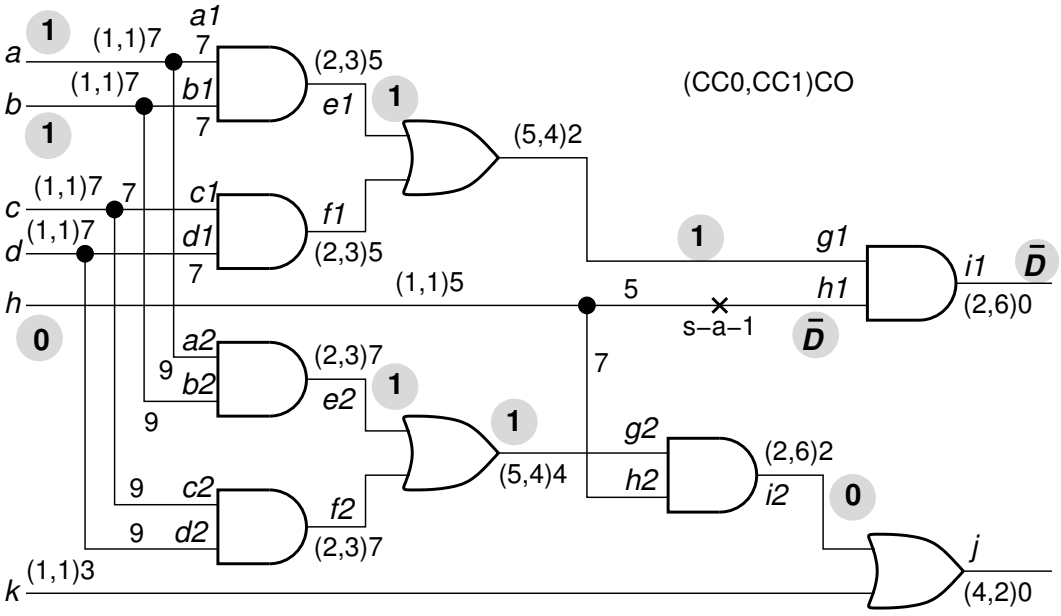
There are no headlines.

Step 1: Goal – sensitize fault
Implication stack – $B = 0$
Implied signals – $B = 0, d = e = f = g = 0, h = l = 0, k = \overline{D}$, $m = n = p = D, q = \overline{D}, r = \overline{D}, s = t = u = D, v = \overline{D}$, $X = 1$, $Y = 1$
D-frontier – Z

Step 2: Goal – propagate fault to Z
Implication stack – $B = 0, C = 1$
Implied signals – $B = 0, C = 0, d = e = f = g = 0, h = l = 0, k = \overline{D}$, $m = n = p = D, q = \overline{D}, r = \overline{D}, s = t = u = D, v = \overline{D}$, $X = 1$, $Y = 1$, $Z = 1$
D-frontier – *empty*
Sensitization and propagation conditions do not intersect. Hence, the fault is proved redundant in two steps without any backtracks.

7.16 FAN

The signal values determined by FAN are shown below. The figure also shows the SCOAP testability measures. In this case there are no headlines.



Steps 1 and 2: Goal – sensitize fault and propagate it to $i1$
Implication stack – $h = 0, g1 = 1$
Implied signals – $g1 = 1, h = 0, i1 = \overline{D}, i2 = 0$
D-frontier – ϕ (fault effect at PO)

Step 3: Goal – set $e1 = 1 \Rightarrow a = 1$ and $b = 1$
Implication stack – $h = 0, g1 = 1, a = 1, b = 1$
Implied signals – $a = 1, b = 1, e1 = 1, e2 = 1, g1 = 1, g2 = 1, h = 0, i1 = \overline{D}, i2 = 0$
D-frontier – ϕ (fault effect at PO)
Goal satisfied, test for fault $h1$ *s-a-1 is* $(a, b, c, d, h, k) = (1, 1, X, X, 0, X)$, $i1 = \overline{D}$*; no backtracks were required.*

7.17 SOCRATES

To obtain a test for the fault n s-a-1 in the circuit of Figure 7.24 (see page 190 of the book and the figure below), we perform static learning:

Signal	Learned implications	Signal	Learned implications
$B = 0$	$(m = 0) \Rightarrow (B = 1)$ $(q = 1) \Rightarrow (B = 1)$ $(r = 1) \Rightarrow (B = 1)$ $(s = 0) \Rightarrow (B = 1)$ $(v = 1) \Rightarrow (B = 1)$	$X = 0$	$(r = 1) \Rightarrow (X = 1)$ $(Y = 1) \Rightarrow (X = 1)$ $(v = 1) \Rightarrow (X = 1)$ $(q = 1) \Rightarrow (X = 1)$ $(s = 1) \Rightarrow (X = 1)$ $(m = 0) \Rightarrow (X = 1)$
$d = 1$	$(m = 0) \Rightarrow (d = 0)$ $(q = 1) \Rightarrow (d = 0)$ $(r = 0) \Rightarrow (d = 0)$ $(s = 1) \Rightarrow (d = 0)$ $(X = 1) \Rightarrow (d = 0)$ $(Y = 1) \Rightarrow (d = 0)$ $(v = 1) \Rightarrow (d = 0)$ $(Z = 0) \Rightarrow (d = 0)$	$Y = 0$	$(X = 1) \Rightarrow (Y = 1)$ $(v = 1) \Rightarrow (Y = 1)$ $(r = 0) \Rightarrow (Y = 1)$ $(d = 0) \Rightarrow (Y = 1)$ $(m = 0) \Rightarrow (Y = 1)$
$Z = 1$	$(q = 1) \Rightarrow (Z = 1)$		

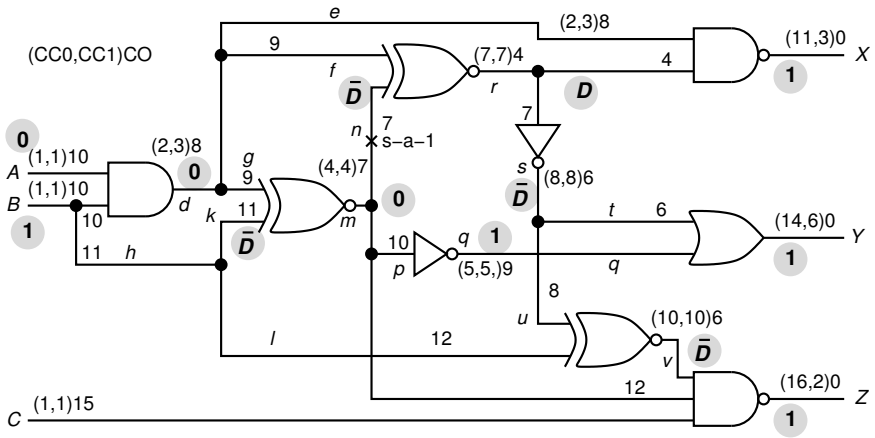
Step 1: Goal – sensitize fault, $m = 0$

Static learning – $B = 1$

Implications – $d = 0, X = 1, Y = 1, A = 0, r = D, q = 1, m = 0, s = \overline{D}$, $v = \overline{D}, Z = 1$

D-frontier – ϕ (null)

Redundant fault, because D-frontier vanishes at gate Z, no decision alternatives. No need for dynamic learning, no use of the constructive dilemma or Modus Tollens. No backtracks.



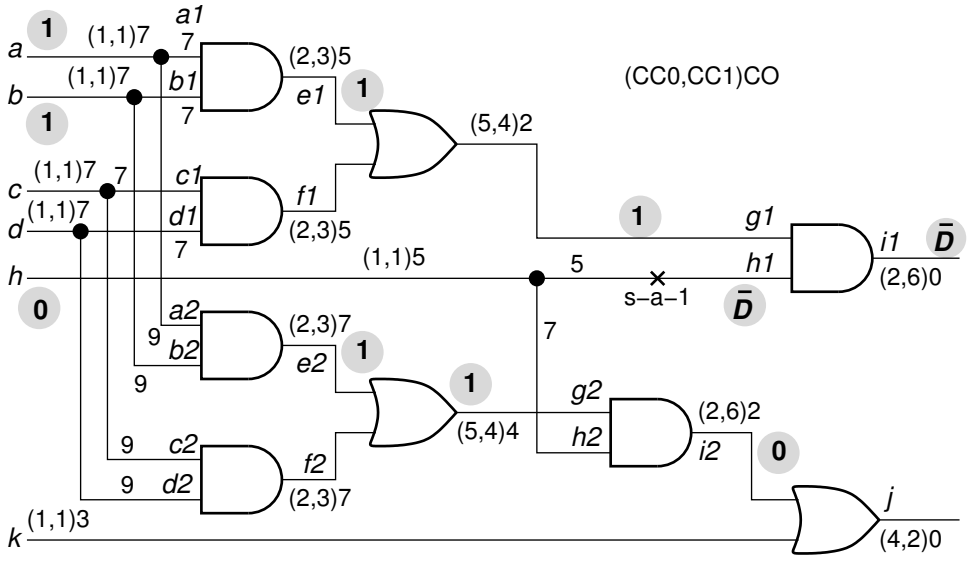
7.18 SOCRATES

To derive a test for the fault $h1$ s-a-1 in the circuit of Figure 7.35 (see page 200 of the book and the figure below), we perform static learning:

Signal	Learned implications
$e1 = 1$	$(e2 = 0) \Rightarrow (e1 = 0)$
$f1 = 1$	$(f2 = 0) \Rightarrow (f1 = 0)$
$e2 = 1$	$(e1 = 0) \Rightarrow (e2 = 0)$
$f2 = 1$	$(f1 = 0) \Rightarrow (f2 = 0)$

Step 1: Goal – sensitize fault
Implication stack – $h = 0$
Implications – $h1 = \overline{D}, i2 = 0$
D-frontier – $i1$
Dynamic learning is not useful.

Step 2: Goal – propagate fault, set $g1 = 1 \Rightarrow e1 = 1$
Implication stack – $h = 0, a = 1, b = 1$
Implications – $h1 = \overline{D}, i2 = 0, g1 = 1, e1 = 1,\ e2 = 1, g2 = 1, i1 = \overline{D}$
D-frontier – ϕ, fault effect at PO
Fault tested in two steps. Constructive Dilemma and Modus Tollens are not useful.
Test is $(a, b, c, d, h, k) = (1, 1, X, X, 0, X)$.

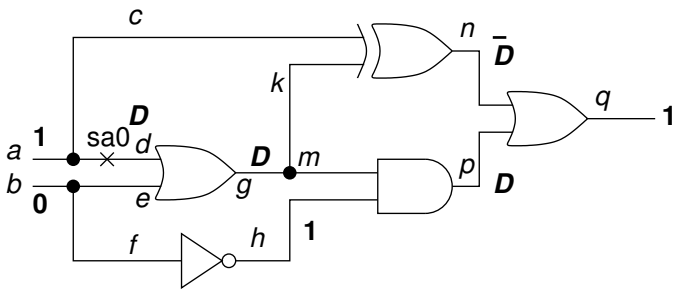


7.19 Redundancy proofs

(1) Proof of d s-a-0 redundant using PODEM.

Step 1: Goal – sensitize fault. Objective – $d = 1$. Backtrace – Implication stack – $a = 1$.
Implication – $d = D$. D-frontier – g.

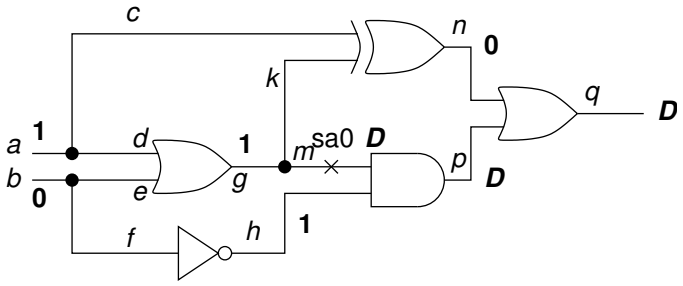
Step 2: Goal – propagate fault. Objective – $e = 0$. Backtrace – Implication stack – $a = 1, b = 0$.
Implications – $d = D, g = D, h = 1, n = \overline{D}, p = D, q = 1$.
D-frontier – ϕ.
Fault proved redundant because D-frontier disappears at q – no alternative assignments possible.



(2) Proof of m s-a-0 testable using PODEM.

Step 1: Goal – sensitize fault. Objective – $g = 1$.
Backtrace – Implication stack – $a = 1$. Implications – $g = 1, m = D, n = 0$.
D-frontier – p.

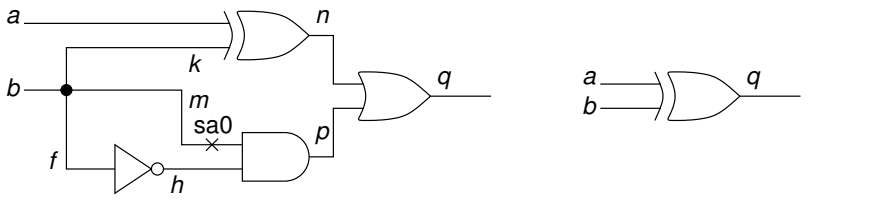
Step 2: Goal – propagate fault. Objective – $h = 1$.
Backtrace – Implication stack – $a = 1, b = 0$. Implications – $g = 1, m = D, n = 0, h = 1, p = D, q = D$.
D-frontier – ϕ *fault at PO.*
Test found – $a = 1, b = 0$*;* $q = D$.



(3) Redundancy removal.

A. Start with redundant fault d s-a-0.

B. Set fault site to the faulty state and find all implications. For $d = 0$, we find $g = b$. Thus, OR gate g is removed and k and m become fanouts of PI b. The *reduced* circuit is shown on the left in the following figure.



Circuit after removing d sa0 fault. Circuit after removing m sa0 fault

C. Examine the reduced circuit for another redundant fault. We find that m s-a-0, which was testable in the original circuit, is now redundant.

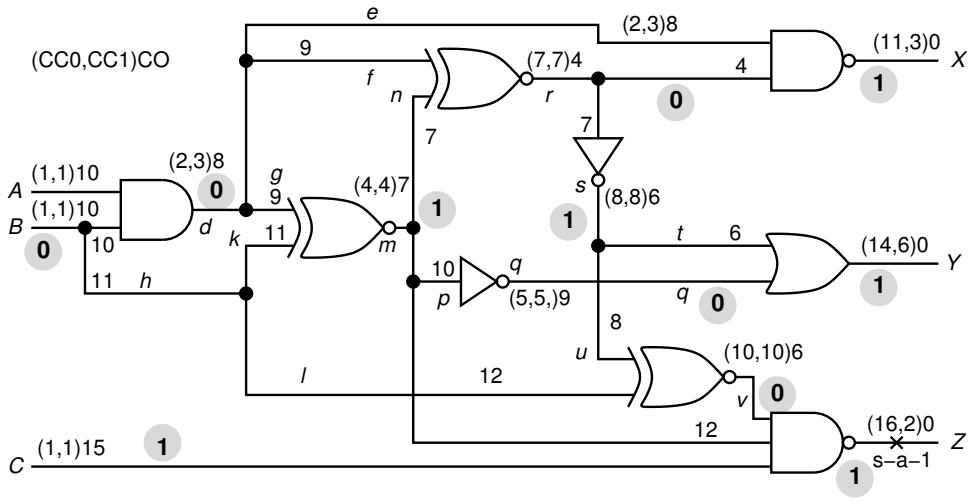
D. Repeat steps B and C until all faults in the reduced circuit are testable.

The above procedure leads to the circuit, $q = a \oplus b$, as shown on the right in the above figure.

Note: This procedure removes only one redundant fault at a time and requires repeated use of ATPG. It is possible to remove several redundant faults together, provided they are selected such that the circuit function is preserved. Removal of a single redundant fault leaves the circuit function unchanged.

7.20 PODEM

SCOAP testability measures for the circuit of Figure 7.24 (page 190 of the book) are shown in the figure below. Steps of PODEM for the fault Z s-a-1 follow.



Step 1: Goal – sensitize fault. Objective – $Z = 0$.
Backtrace – Implication stack – $C = 1$. Implications – none. D-frontier – empty.

Step 2: Objective – $Z = 0$.
Backtrace – Implication stack – $C = 1, B = 1$ ($m = 1 \Rightarrow B = 1$).
Implications – none. D-frontier – empty.

Step 3: Objective – $Z = 0$.
Backtrace – Implication stack – $C = 1, B = 1, A = 1$
($m = 1 \Rightarrow d = 1 \Rightarrow A = 1$).
Implications – $d = 1$, $m = 1$, $q = 0$, $r = 1$, $s = 0$, $v = 0$, $Z = 1$, $X = 0$, $Y = 0$.
Objective cannot be met with these inputs.

Step 4: Objective – $Z = 0$.
Backtrack – Implication stack – $C = 1, B = 1, A = 0$.
Implications – $d = 0$, $m = 0$, $q = 1$, $r = 1$, $s = 0$, $v = 0$, $Z = 1$, $X = 1$, $Y = 1$.
Objective cannot be met with these inputs.

Step 5: Objective – $Z = 0$.
Backtrack – Implication stack – $C = 1, B = 0$.
Implications – $d = 0$, $m = 1$, $q = 0$, $r = 0$, $s = 1$, $v = 0$, $Z = 1$, $X = 1$, $Y = 1$.
Objective cannot be met with these inputs.

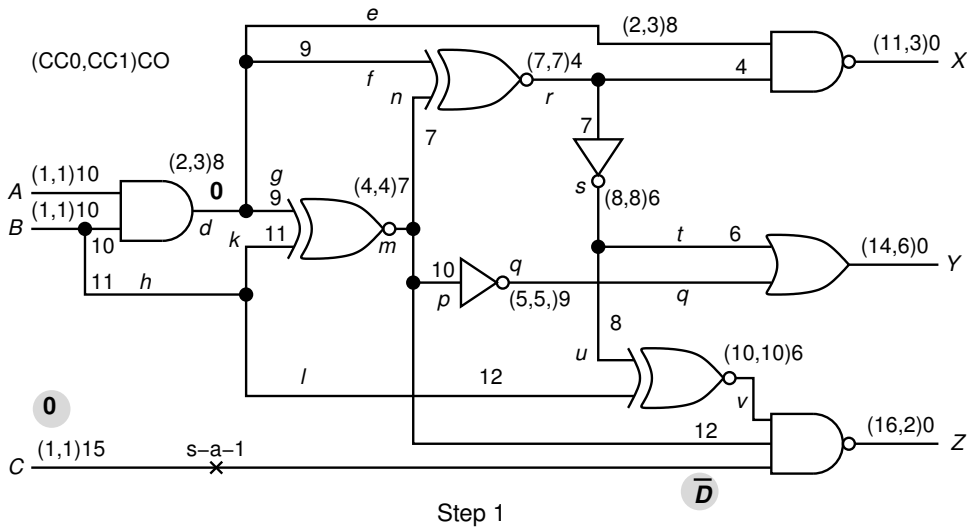
Step 6: Objective – $Z = 0$.
Backtrack – Implication stack – $C = 0$.
Implications – $Z = 1$.
Objective cannot be met with this input.

Step 7: Objective – $Z = 0$.
Backtrack – Implication stack – *empty.*
No other input choices left, test is impossible, *fault is redundant.*
4 backtracks required.

7.21 SOCRATES

The SOCARTES solution for fault C s-a-1 in the circuit of Figure 7.24 (page 190 of the book) we will use static learning given in the solution to Problem 7.17. The algorithm proceeds as follows:

Step 1: Objective – set $C = 0$.
Implication stack – $C = 0$.
Implications – none.
D-frontier – Z.
The next figure shows the result of this step.



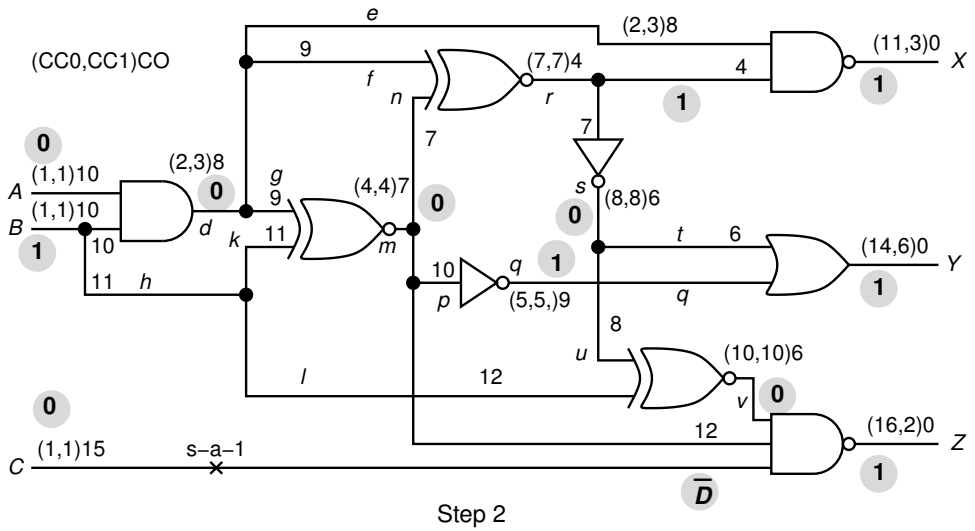
Step 2: Objective – set $v = 1$.

Implication stack (static learning) – $C = 0, B = 1$.

Implications (static learning) – $d = 0$, $X = 1$, $A = 0$, $m = 0$, $q = 1$, $Y = 1$, $r = 1$, $s = 0$, $v = 0$, $Z = 1$.

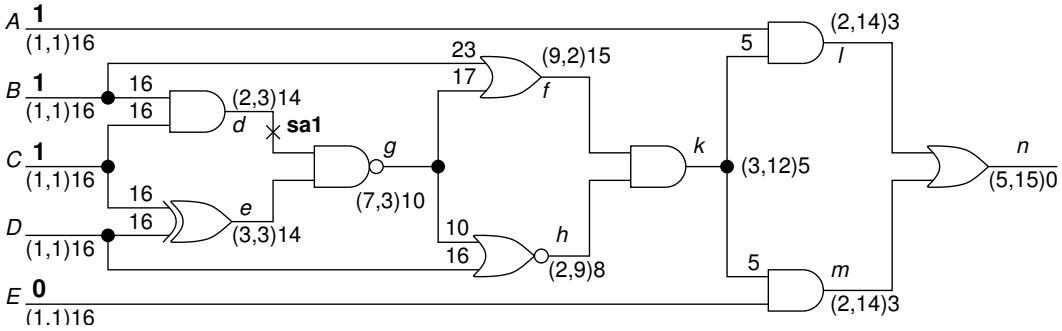
D-frontier – ϕ (null).

Propagation blocked, no alternative choices, *fault found to be redundant with 0 backtracks.* Dynamic learning, Constructive Dilemma and Modus Tollens not used. The following figure shows step 2.



7.22 FAN

Test generation by FAN for the fault d s-a-1 in the circuit of Figure 7.30 (see page 197 of the book and the figure below) is as follows. We identify k as a headline.



Step 1: Objective – propagate any fault effect from k to n.
Implication stack – $A = 1$.
Implications – $l = D$ or $\overline{D}$.
D-frontier – n.

Step 2: Objective – propagate any fault effect from l to n.
Implication stack – $A = 1$, $E = 0$.
Implications – $l = D$ or $\overline{D}$, $m = 0$, $n = D$ or $\overline{D}$.
D-frontier – $-$.

Step 3: Objective – sensitize fault.
Implication stack – $A = 1$, $E = 0$, $C = 0$.
Implications – $d = \overline{D}$, $l = D$ or $\overline{D}$, $m = 0$, $n = D$ or $\overline{D}$.
D-frontier – g.

Step 4: Objective – propagate fault to k, set $e = 1$.
Implication stack – $A = 1$, $E = 0$, $C = 0$, $D = 1$.
Implications – $d = \overline{D}$, $e = 1$, $g = D$, $h = 0$, $k = 0$, $l = 0$, $m = 0$, $n = 0$.
D-frontier – ϕ; D-frontier disappeared, so backtrack.

Step 5: Objective – propagate fault to k,
Implication stack – $A = 1$, $E = 0$, $C = 0$, $D = 0$.
Implications – $d = \overline{D}$, $e = 0$, $g = 1$, $h = 0$, $k = 0$, $l = 0$, $m = 0$, $n = 0$.
D-frontier – ϕ; D-frontier disappeared, so backtrack again.

Step 6: Objective – sensitize fault.
Implication stack – $A = 1$, $E = 0$, $C = 1$, $B = 0$.
Implications – $d = \overline{D}$, $l = D$ or $\overline{D}$, $m = 0$, $n = D$ or $\overline{D}$.
D-frontier – g.

Step 7: Objective – propagate fault to k.
Implication stack – $A = 1$, $E = 0$, $C = 1$, $B = 0$, $D = 0$.

Implications – $d = \overline{D}$, $e = 1$, $g = D$, $f = D$, $h = \overline{D}$, $k = 0$, $l = 0$, $m = 0$, $n = 0$.
D-frontier – ϕ; D-frontier disappeared, so backtrack.

Step 8: Objective – propagate fault to k.
Implication stack – $A = 1$, $E = 0$, $C = 1$, $B = 0$, $D = 1$.
Implications – $d = \overline{D}$, $e = 0$, $g = 1$, $f = 1$, $h = 0$, $k = 0$, $l = 0$, $m = 0$, $n = 0$.
D-frontier – ϕ; D-frontier disappeared, so backtrack again.

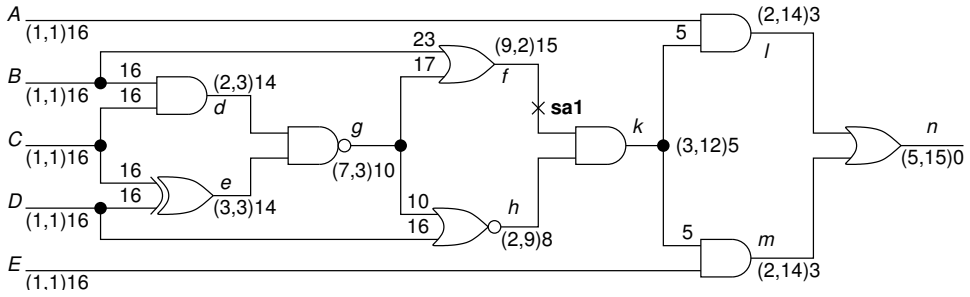
Step 9: Objective – sensitize fault.
Implication stack – $A = 1$, $E = 0$, $C = 1$, $B = 1$.
Implications – $d = 1$, $l = D$ or $\overline{D}$, $m = 0$, $n = D$ or $\overline{D}$.
D-frontier – –; D-frontier disappeared, so backtrack, but no alternatives left.
Fault is redundant (4 backtracks.)

7.23 SOCRATES

For a SOCRATES solution to find a test for the fault f s-a-1 in the circuit of Figure 7.30 (see page 197 of the book and the next figure) we use static learning:

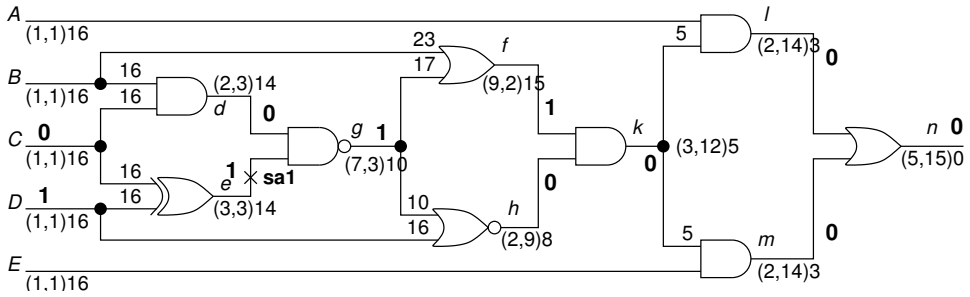
Signal	Learned implications	Signal	Learned implications
$B = 0$	$(n = 1) \Rightarrow (B = 1)$	$k = 1$	$(h = 0) \Rightarrow (k = 0)$
$C = 0$	$(n = 1) \Rightarrow (C = 1)$		$(g = 1) \Rightarrow (k = 0)$
$D = 1$	$(n = 1) \Rightarrow (D = 0)$		$(d = 0) \Rightarrow (k = 0)$
$d = 0$	$(n = 1) \Rightarrow (d = 1)$		$(e = 0) \Rightarrow (k = 0)$
$e = 0$	$(n = 1) \Rightarrow (e = 1)$	$l = 1$	$(k = 0) \Rightarrow (l = 0)$
$g = 0$	$(k = 0) \Rightarrow (g = 1)$		$(h = 0) \Rightarrow (l = 0)$
$g = 1$	$(n = 1) \Rightarrow (g = 0)$		$(g = 1) \Rightarrow (l = 0)$
$f = 0$	Redundant, f can never be 0		$(d = 0) \Rightarrow (l = 0)$
$h = 0$	$(n = 1) \Rightarrow (h = 1)$		$(e = 0) \Rightarrow (l = 0)$
$h = 1$	$(k = 0) \Rightarrow (h = 0)$	$m = 1$	$(k = 0) \Rightarrow (m = 0)$
	$(g = 1) \Rightarrow (h = 0)$		$(h = 0) \Rightarrow (m = 0)$
	$(e = 0) \Rightarrow (h = 0)$		$(g = 1) \Rightarrow (m = 0)$
	$(d = 0) \Rightarrow (h = 0)$		$(d = 0) \Rightarrow (m = 0)$
$k = 0$	$(n = 1) \Rightarrow (k = 1)$		$(e = 0) \Rightarrow (m = 0)$

Step 1: Objective – sensitize fault, set $f = 0$.
Static learning indicates that f can never be 0.
Redundant fault, found with 0 backtracks.



7.24 PODEM

A PODEM solution for a test for the fault e s-a-1 in the circuit of Figure 7.30 (page 190 of the book and the next figure) is as follows:



Step 1: Objective – sensitize fault, $(e = 0) \rightarrow (C = 1)$.
Implication stack – $C = 1$.
Implications – none.
D-frontier – null.

Step 2: Objective – sensitize fault, $(e = 0) \rightarrow (D = 1)$.
Implication stack – $C = 1$, $D = 1$.
Implications – $e = \overline{D}$, $h = 0$, $k = 0$, $l = 0$, $m = 0$, $n = 0$.
D-frontier – null; since D-frontier vanishes, backtrack.

Step 3: Objective – sensitize fault.
Implication stack – $C = 1$, $D = 0$.
Implications – $e = 1$.
D-frontier – null; fault not sensitized, backtrack.

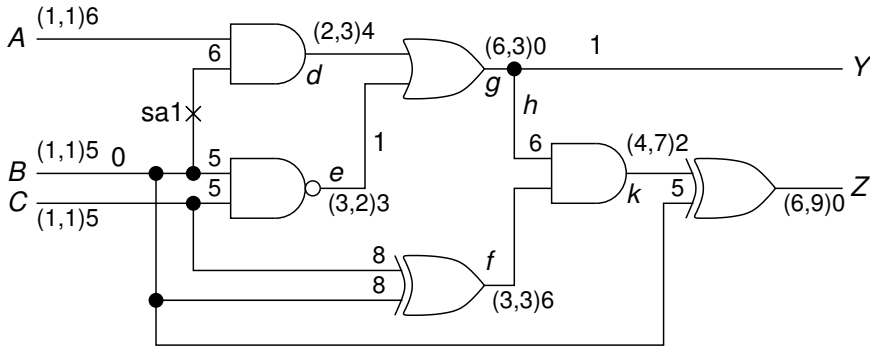
Step 4: Objective – sensitize fault, $(e = 0) \rightarrow (D = 0)$.
Implication stack – $C = 0$, $D = 0$.
Implications – $e = \overline{D}$, $d = 0$, $g = 1$, $f = 1$, $h = 0$, $k = 0$, $l = 0$, $m = 0$, $n = 0$.
D-frontier – null; since D-frontier vanishes, backtrack.

Step 5: Objective – sensitize fault.
Implication stack – $C = 0$, $D = 1$.

Implications – $e = 1$, $d = 0$, $g = 1$, $f = 1$, $h = 0$, $k = 0$, $l = 0$, $m = 0$, $n = 0$.
D-frontier – null; fault not sensitized, no choices left, *fault is redundant*, found with 3 backtracks.

7.25 PODEM

PODEM solution for a test for the fault $B - d$ s-a-1 in the circuit of Figure 7.39 (page 207 and the figure below) is as follows:



Step 1: Objective – sensitize fault.
Implication stack – $B = 0$.
Implications – $e = 1$, $g = 1$.
D-frontier – null.
X-path check indicates no D-frontier. No alternative decisions possible. *Redundant fault*, found with no backtracks.

7.26 Static compaction

Forward order	Reverse order
$t_1 \cap t_2 = 1010$	$t_5 \cap t_4 = 1100$
$t_3 \cap t_4 = 0100$	$t_3 \cap t_1 = 0010$
$t_5 = 1100$	$t_5 \cap t_4 \cap t_2 = 1100$

Compacted vector sets:
Forward order compaction: 1010, 0100, 1100
Recerse order compaction: 0010,1100
Reverse order is better, as it gives 1 fewer vectors.

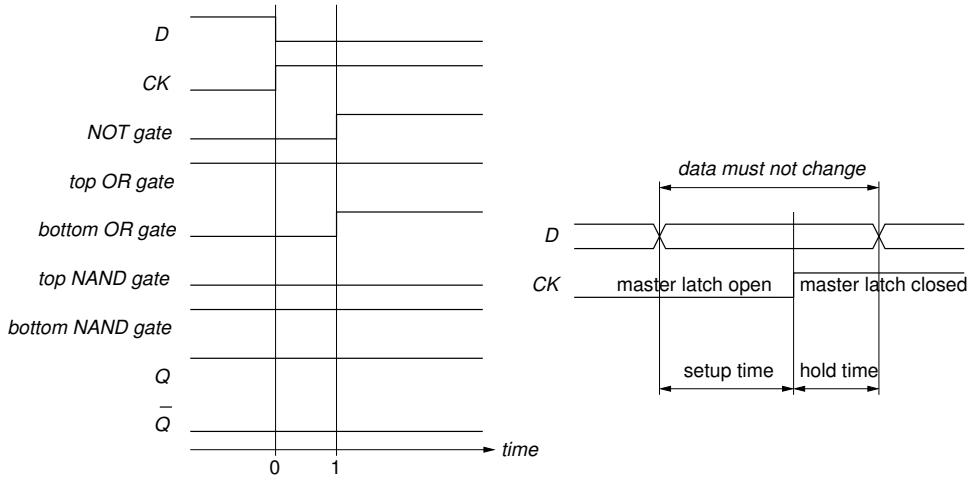
Chapter 8: Sequential Circuit ATPG

8.1 Race condition

The signals are sketched in the timing diagram below. We assume ideal logic signals that change at times 0, 1, 2, etc.:

- $time = 0$, D falls and CK rises.
- $time = 1$, outputs of NOT gate and the bottom OR gate rise.
- $time \geq 1$, all signals retain their values without any further change.

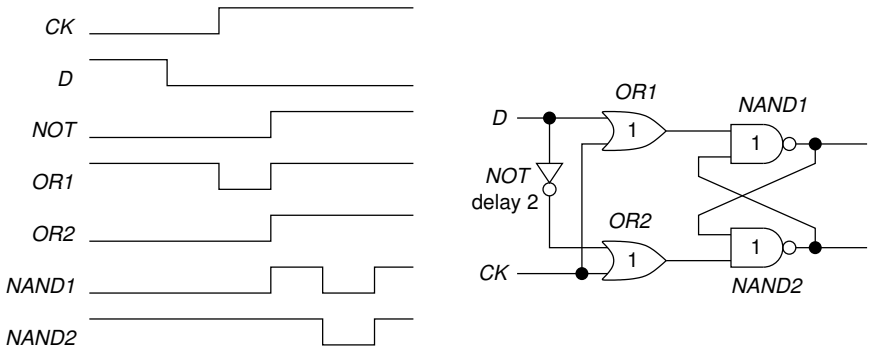
Neither the state of the master latch nor that of the slave latch is affected by the change in D. To be stored correctly in the flip-flop, the data input (D) should change earlier than the rising edge of CK by an interval known as the *setup time*. Also, the data should remain unchanged beyond the rising edge of CK for a duration known as the *hold time*.



Setup time is the time for the master latch to acquire a steady state after the D input changes while the clock is in the active state (0 for the flip-flop of Figure 8.2 in the book.)

Hold time is the delay of the clock control gates (OR gates in the flip-flop of Figure 8.2.) It is the interval that the clock takes to isolate the storing gates (two NAND gates) of the master latch from the data input.

In the above case, data and clock changed simultaneously and the flip-flop recorded the wrong (old) data. We illustrate a peculiar behavior of the latch when data and clock changes occur close to each other. As shown in the next figure, suppose the NOT gate has a delay of two units and all other gates have one unit of delay. Suppose CK rises one unit after the fall of D. This produces simultaneous $0 \rightarrow 1$ transitions at the outputs of the two OR gates. The two equal delay NAND gates now oscillate between 00 and 11 states.



The oscillations we observe in this example do not actually occur. Any unbalance in the delays of the NAND gates will stabilize the state of the latch to either 01 or 10 state. Such delay-dependent behavior is commonly known as the *race condition* or *metastability*. In our example, a race is possible if the separation between the clock and data transitions is less than the delay of NOT gate. In general, a race condition or metastability is avoided if the setup and hold time restrictions are satisfied.

8.2

It requires just one vector to initialize the circuit. If the initial state is unknown, i.e., $C_n = X$, the vector $A_n = B_n = 1$ initializes the state to 1, irrespective of the presence of any fault at the output S_n. Given this state, detection of any output fault at the output reduces to a combinational ATPG problem of setting the output to the opposite value. This can be done by a single vector: $(A_n = 0, B_n = 0)$ will set the output to 1 or $(A_n = 0, B_n = 1)$ will set it to 0. Thus, just two vectors, an initialization vector 11 followed by an appropriate vector to set the output, will detect the output fault in the circuit of Figure 8.3 (see page 215 of the book.)

8.3

Considering the combinational logic of the circuit we find that for sensitizing a path from a PI to PO, S_n, we must specify the other PI as well as the present state, C_n. Thus, the circuit must be first initialized. Any input fault in the circuit of Figure 8.3 (see page 215 of the book) can be tested as follows:

Vector 1 (Initialization.) If the fault is s-a-1 type, then vector 11 is used to initialize the circuits (both good and faulty) to 1. If the fault is s-a-0 type, then vector 00 initializes the circuits to 0.

Vcetor 2 (Fault activation and path sensitization.) For a s-a-1 fault, the circuit has been initialized to a 1 state. A 0 is applied to the faulty line, activating the fault as 0/1. Application of 1 to the other input propagates a value 0/1 to the output S_n. For a s-a-0 fault, the circuit is initialized to a 0 state. An input vector 11 now activates the fault and also propagates its effect to S_n.

Thus, only two vectors are needed to test any input fault.

8.4

The required test has two steps:

1. *Fault activation.* Assuming the present state to be unknown, we set the next state to 1. For $C_n = X$, backward justification of $C_{n+1} = 1$ in Figure 8.3 (see page 215 of the book) gives us $A_n = 1$ and $B_n = 1$.

2. *Path sensitization.* For the next vector, the above next state becomes the present state and the fault C_n s-a-0 is sensitized. We sensitize a path from C_n to S_n by setting $A_n = 1$ and $B_n = 1$.

Thus, the test sequence is (A_n, B_n) = (1,1), (1,1).

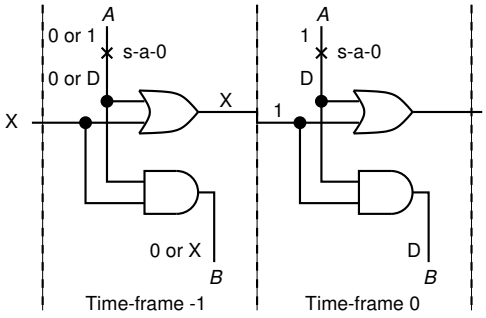
8.5

For test generation with the five-valued algebra, we use the following steps (also see the illustration):

Step 1: Place a D at the output B in time-frame 0.

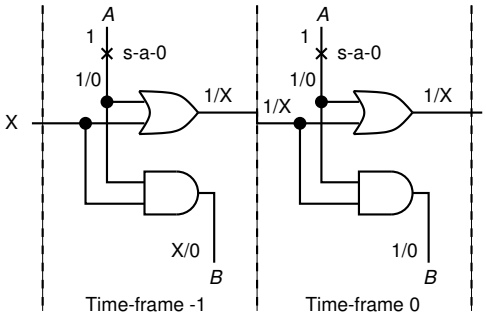
Step 2: This can only be justified by either DD or D1 input to the AND gate in time-frame 0. DD is not possible due to the state input being X in the time-frame -1. We place D1 by applying $A = 1$ and assuming that a state 1 can be justified.

Step 3: Any input, 0 or 1, as shown in the figure, produces a state output X from time-frame -1. Thus, the faulty circuit cannot be initialized to any known state, including the 1 needed for the test. **Hence, it is impossible to find a test by the 5-valued algebra.**



Test generation attempted with 5-valued algebra.

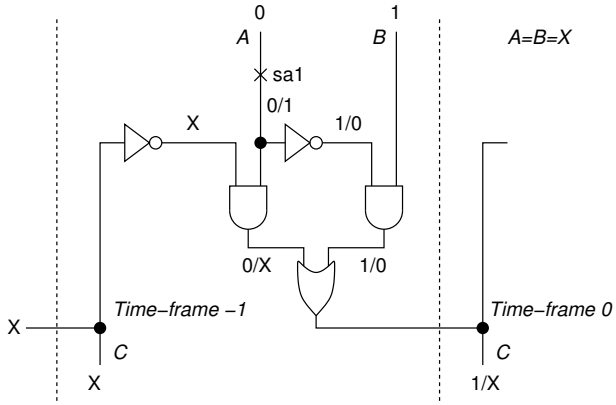
Following similar steps with the nine-valued algebra (see illustration below), we find that two 1's at A detect the fault at B as 1/0 in time-frame 0. Notice that the fault is detected although the faulty circuit is never initialized.

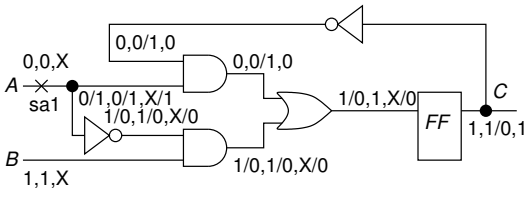


Test generation with 9-valued algebra.

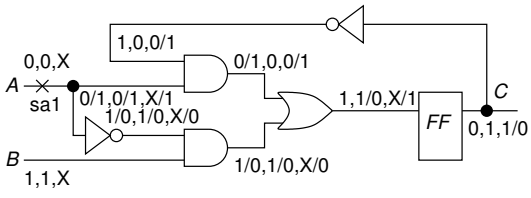
8.6 Initialization fault

The following figure illustrates the time-frame expansion procedure of generating a vector, $A = 0,\ B = 1$, which starting from the unknown state detects the fault A s-a-1 as $1/X$. After the application of the input vector, the flip-flop is clocked before the output can be observed. Even if we add more vectors to the test sequence, the faulty circuit output will not become deterministic. This is because the faulty circuit is not initializable. The fault is only potentially detectable.





Test simulation with initial state 1.



Test simulation with initial state 0.

Note: Some test generators will find the potential detection test of the above type. Others will consider the fault untestable (conservative approach.) Most fault simulators will find the fault potentially detectable. Interestingly, the two test simulation scenarios in the figure show that the fault is definitely detectable, though the detection requires multiple observations. If we assume the initial state to be 1 then the fault is detected as 1/0 after the application of the first clock. However, this

output will be 1 (same as the correct output) if the initial state was 0. In this case, repeating the same vector and clocking once again will produce a 1/0 output. A conventional fault simulator will not report such detection because it does not enumerate the possible initial state scenarios. For such multiple observation tests see reference [525] of the book.

8.7

The *note* in the solution of Problem 8.6 explains the operation of a multiple observation test. Besides simulation, a multiple observation test can also be derived by the following procedure.

An observable state variable, which cannot be initialized in the faulty circuit but must be observed for fault detection, is represented symbolically by a Boolean variable s. Inversion of s is $\overline{s}$. A test sequence is derived such that any one of the following pairs of outputs is produced:

- $0/s$ and $0/\overline{s}$
- $1/s$ and $1/\overline{s}$
- $0/s$ and $1/s$
- $0/\overline{s}$ and $1/\overline{s}$

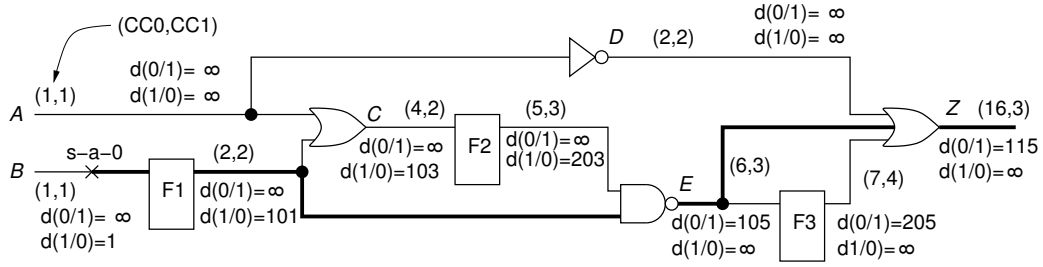
We notice that irrespective of the value the uninitialized state variable assumes, one element in each test output pair will provide definite fault detection. For example, the outputs produced by the test (A, B) = (0,1), (0,1) of Problem 8.6 are $1/\overline{s}$ and $1/s$, respectively, which agree with the second pair given above.

When the feedback in the circuit of Figure 8.25 (see page 250 of the book) has no inversion, a test sequence (A, B) = (0,0), (0,1) will produce outputs $0/s$ and $1/s$. This is a multiple observation test. Details on multiple observation tests may be found in reference [525] cited in the book.

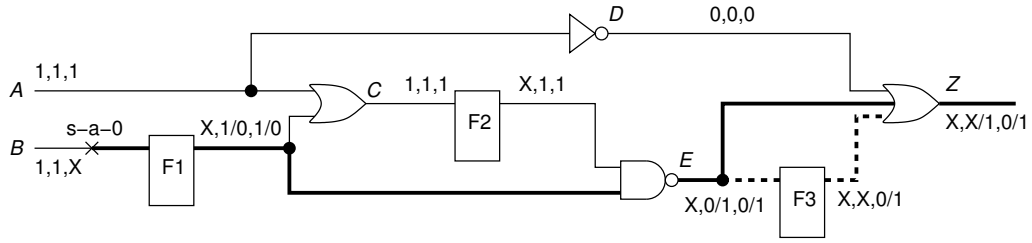
8.8

The following figure shows the combinational 0 and 1 controllabilities as $(CC0, CC1)$. Notice that the output measures for a flip-flops are obtained by just adding 1 to the input measures. This is due to assumptions that the clock has controllabilities (1,1) and the combinational depth of a flip-flop is 0. The fault site can be driven to 1/0 by controlling $B = 1$ and it cannot be driven to 0/1. Thus, its drivabilities are $d(0/1) = \infty$ and $d(1/0) = 1$, respectively. Drivabilities of all other signals are successively computed by simple path sensitization.

The path shown in bold lines is the least drivability (minimum effort) path. A test obtained by a drivability-based ATPG procedure is shown in the lower figure. This three-vector test, $(A, B) = (1, 1), (1, 1), (1, X)$, sensitizes the minimum drivability path and we find that another path, shown by dotted lines, must also be sensitized.



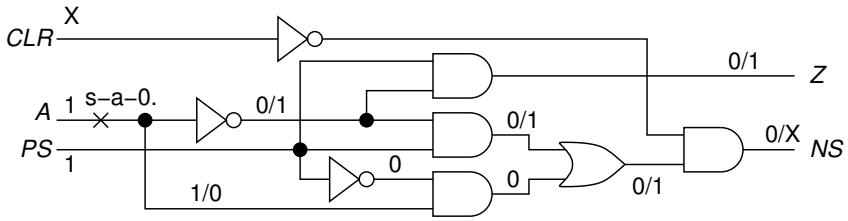
Drivabilities for fault B s–a–0 in circuit of Figure 8.9. Bold lines show easiest drivability path.



A three–vector test for fault B s–a–0. Dotted lines show an additional path sensitized.

8.9 Approximate test

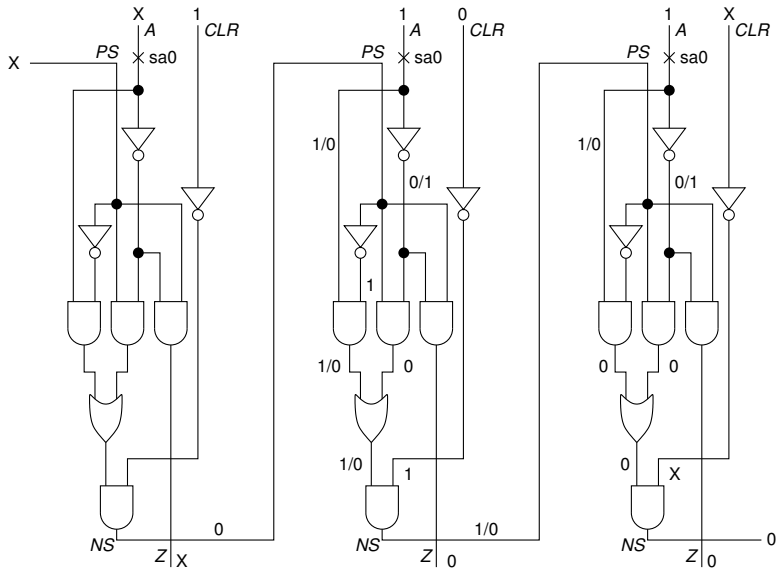
A combinational test for the fault A s-a-0, as shown in the following figure, is $CLR = X,\ A = 1,\ PS = 1$. The fault is detected at Z as $0/1$.



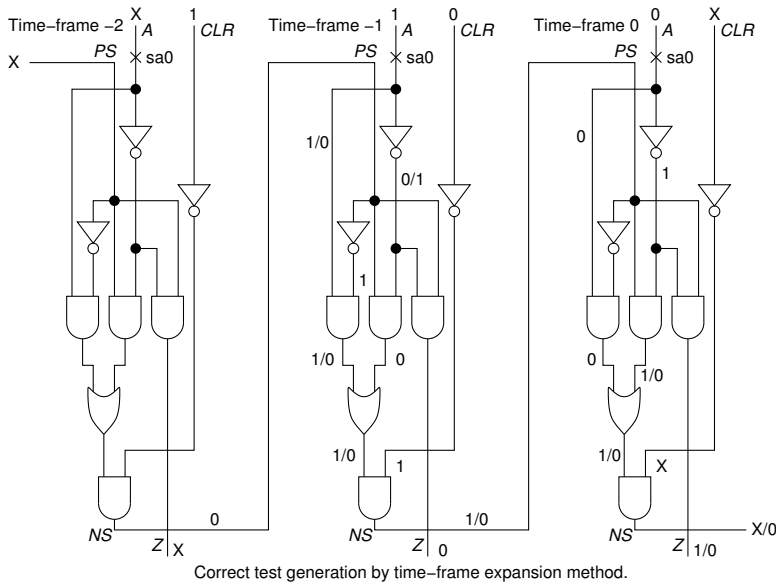
Combinational test for A s–a–0.

To justify $PS = 1$ in this test, we generate an input vector for the combinational circuit that will produce $NS = 1$ output. We find a vector, $CLR = 0,\ A = 1,\ PS = 0$. In order to apply the required approximation, we assume no fault during justification. The justification must continue until we can find a vector with $PS = X$. $PS = 0$ is easily justified by an input, $CLR = 1,\ A = X,\ PS = X$. Thus, the test sequence contains three vectors, $(CLR, A, PS) = (1, X, X)$, $(0, 1, 0)$, $(X, 1, 1)$, which is simulated in the next figure. We find that the test fails to detect the fault. In the last time-frame, where the combinational vector is applied, the PS input is $1/0$ instead of 1. This is due to the fault being present in the previous time-frame. Thus the faulty previous state interferes with the newly generated fault effect and the output Z becomes 0 instead of $0/1$.

A valid test is generated by time-frame expansion when the fault is assumed to be present in all time-frames (as we did for simulation in the above figure.) The new



test, as shown in the following figure, has only one change. In the last time-frame A is changed to 0. So, no new fault effect is produced there and the fault effect 1/0 produced in time-frame -1 is propagated to Z.



The test sequence is $(CLR, A, PS) = (1, X, X)$, $(0, 1, 0)$, $(X, 0, 1/0)$.

8.10

A necessary condition for detection of a fault in a sequential circuit is that there must exist at least one time-frame in which,

1. the fault is activated, and
2. the fault effect is propagated to the boundary of the combinational logic, i.e., to one or more PO and/or one or more state variables.

Since the fault is combinationally untestable it is impossible to satisfy these conditions even though the state inputs are assumed to be fully controllable. Thus, no vector sequence can be generated to test the fault in the sequential circuit. ■

8.11

Consider the time-frame expansion method of sequential circuit ATPG. A time-frame consists of combinational logic with some fault activity (fault activation and path sensitization.) In general, this activity must be justified at the PIs of the time-frame by three-valued (0, 1 and X) logic and at the state inputs by nine-valued (0, 1, 0/1, 0/X, . . etc.) logic.

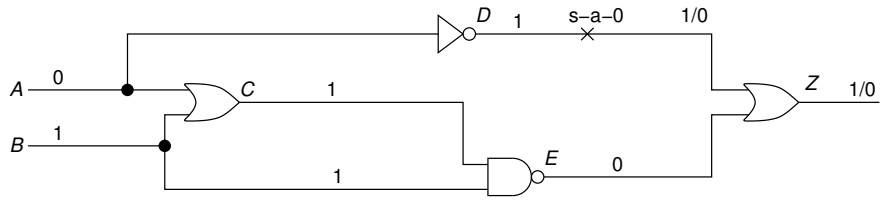
There are two types of time-frames, ones in which the fault is activated, and others where the fault is not activated. Let us consider the time-frame in which the fault is activated for the first time. To be a part of the test sequence, this time-frame must propagate the fault effect either to a PO or to a state variable. We call this the "first detection time-frame." Clearly, such a time-frame is necessary for fault detection.

In the first detection time-frame a combinational test detects the fault at its boundary (PO or state output) when a suitable test vector at PI and state inputs is applied. All preceding time-frames then only generate fault-free states leading to a state input that is necessary for the first detection time-frame. If the combinational test cannot be justified then the first detection time-frame will be impossible and no sequential test can be obtained for the targeted fault. ■

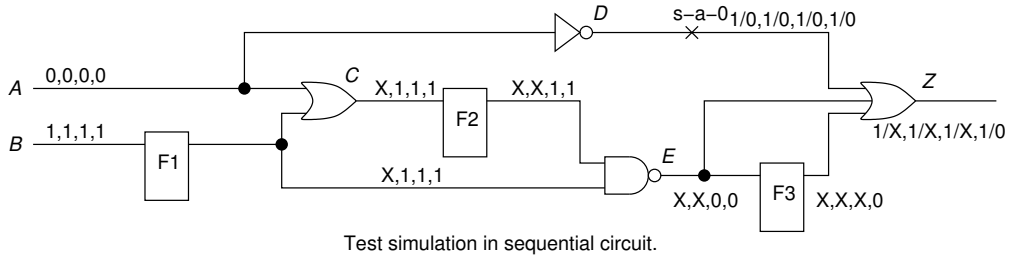
A more detailed discussion of this result may be found in the reference [30] cited in the book.

8.12 Pseudo-combinational test

The pseudo-combinational circuit and a combinational test, $A = 0,\ B = 1$, for the fault D s-a-0 are shown in the following figure. Simulation of the sequential circuit with input $A = 0,\ B = 1$, repeated four times shows that the fault will be detected as 1/0 appearing as the fourth output. We assume that the initial states of all three flip-flops are X.



Pseudo–combinational circuit for the sequential circuit of Figure 8.9..



Test simulation in sequential circuit.

8.13

A pseudo-combinational circuit is obtained by shorting all flip-flops in an acyclic synchronous sequential circuit. We will prove that a test vector for the former, when repeated $d_{seq} + 1$ times, will be a test sequence for the latter, where d_{seq} is called the sequential depth and is the maximum number of flip-flops in any input to output path. Our proof is based on a series of observations:

Observation 1: A clocked flip-flop is equivalent to a delay that equals the clock period, T.

Observation 2: The output of a combinational circuit with arbitrary delays is uniquely determined by the input vector provided (a) output is allowed to stabilize through a time interval, which equals the longest input to output combinational path delay after the input is applied, and (b) the input is held constant throughout that time interval.

Observation 3: A combinational circuit with a single stuck-at fault (and many other non-feedback types of faults) is also a combinational circuit.

Observations 1 and 2 specify that the basic difference between an acyclic sequential circuit and its pseudo-combinational circuit is the delay. The delay of the former has an upper bound, $(d_{seq} + 1)T$, where T is the clock period. The delay of the latter equals that of the longest combinational path in that circuit. Note that T is greater than the longest combinational path delay.

The given test vector produces two different outputs from the good and faulty pseudo-combinational circuits. If the conditions of Observation 1 are satisfied, then the good and faulty acyclic sequential circuits will produce outputs that will differ in a similar way. This is done by holding the vector at the input for an interval $(d_{seq} + 1)T$ and clocking the circuit d_{seq} times. ■

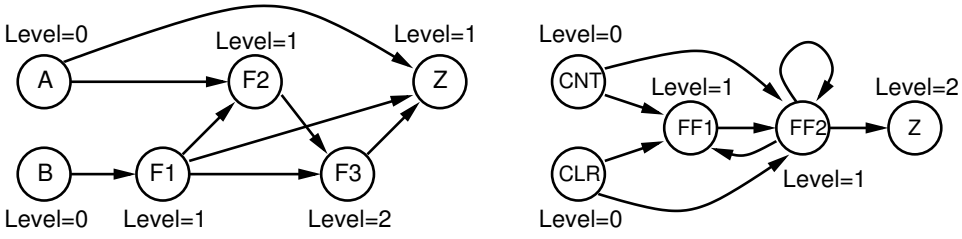
8.14

"A circuit is initializable" means, given that all flip-flops are in unknown (X) states, there exists a finite-length input sequence that will bring all flip-flops to known states. Initializability is often considered in a narrower (and practical) sense to mean that the finite-length sequence, when simulated by a three-valued logic simulator, will set all flip-flops in deterministic (0 or 1) states.

The required proof follows from contradiction. We begin with an assertion that an uninitializable circuit is cycle-free. Then its s-graph is a directed acyclic graph (DAG), which can be levelized according to the maximum distance from PIs. Levels of flip-flop vertices must be contiguous integers from 1 to d_{seq}, the sequential depth. All flip-flops in level 1 are controlled by PIs and can be set to some (may not be every) known states by one input vector followed by a clock. Similarly, all flip-flops in level 2 are controlled by PIs and the flip-flops of level 1 (which are now in known states) and these can be set to known states by a second input vector followed by another clock. Following this procedure, by the time d_{seq} input vectors have been applied, each followed by a clock, all flip-flops will be in known states. Since, d_{seq} for a DAG is a finite integer, the circuit is initialized by a finite length input sequence. This contradicts our assertion. Hence, the circuit cannot be cycle-free and must be cyclic. ■

8.15 Cyclic circuits

Modified s-graphs with PI and PO vertices are shown below. The levels shown give the minimum distance from PIs. The depths of the two circuits are 1 and 2, respectively. This depth gives a lower bound on the length of a test sequence for a fault. In practice, however, a test sequence is almost always longer than this lower bound. The maximum distance levelization and the corresponding depth is a more realistic measure of the test length for a cycle-free circuit. For cyclic circuits no tight measure of test length exists. For an upper bound of $9^{N_{ff}}$ on the test length, where N_{ff} is the number of flip-flops in the circuit, see Section 8.2.5 of the book.



Cycle–free circuit of Figure 8.9. Cyclic circuit of Figure 8.13.

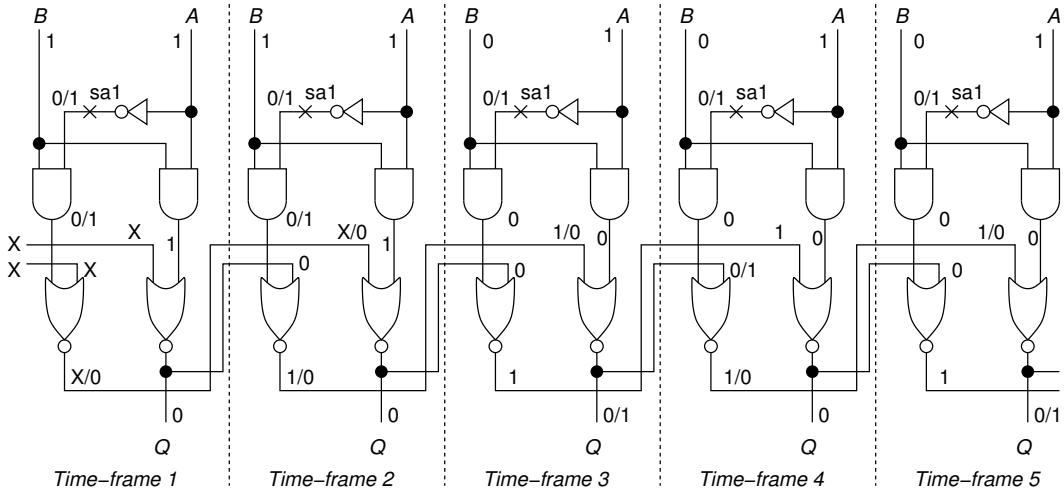
Minimum distance levelization of s–graphs.

8.16 Race fault in asynchronous circuit

A procedure to test the s-a-1 fault at the output of the NOT gate in the circuit of Figure 8.27 is outlined below:

1. We inject the values of A and $\overline{A}$ into the feedback loop consisting of the two NOR gates by applying $B = 1$. $A = 1$ is applied to activate the fault. We assume that the two NOR gates have equal delays and simulate their outputs independently, with the feedback inputs in the unknown (X) state. This is illustrated in time-frame 1 in the following figure.

2. The outputs of NOR gates are applied after the feedback delays in time-frame 2. We find that the outputs, 1/0 and 1, are stable since another time-frame will not change them.



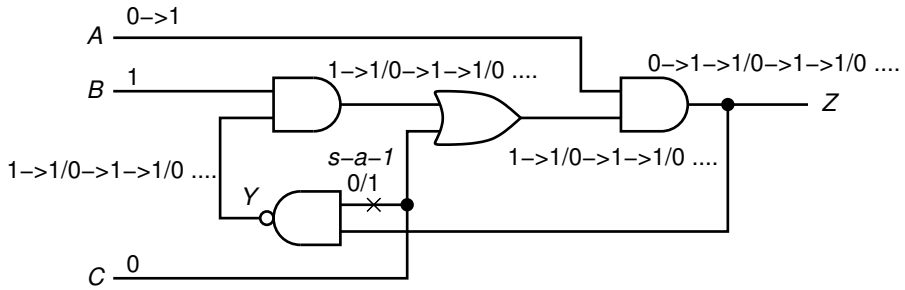
3. Next we apply $B = 0$ to activate the loop. Time-frames 3 through 5 show that in the good circuit the Q output stabilizes to state 0 and the output of the other NOR gate stabilizes to 1. In the faulty circuit, the outputs of the two NOR gates oscillate as 11, 00, 11, . . . This oscillation in the idealized logic model is a manifestation of a *metastable* behavior. The output Q may settle to a 1 or to a 0 state depending upon the relative delays of the two NOR gates. In the absence of more detailed knowledge of circuit parameters (delays, etc.) we consider the fault to be potentially detectable.

Note: Some ATPG programs will consider this fault to be untestable. Strictly speaking, the logic model does not have the information to find tests for such faults, which are often classified as race faults. The "race" refers to an unstable equilibrium in which two possible states compete, each trying to win by getting through the feedback path first. When dealing with the analog behavior of the circuit, this condition is referred to as metastability. For some set of gate delays the circuit will settle in the correct state and the fault would be considered redundant. For other delays the

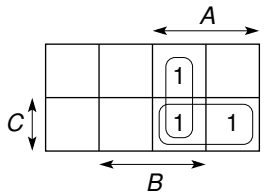
output will settle in the wrong state and the circuit, which is then indeed faulty, will be found to be so by the test.

8.17 Oscillation fault

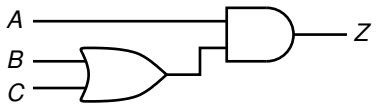
Let us denote the output of NAND gate as Y. The following figure shows test generation using nine-value logic. First, we initialize $Z = 0$ and $Y = 1$ by setting $A = 0$. C is then set to 0 to activate the fault as 0/1. To propagate the faulty state to Y, Z is set to 1 by applying $A = B = 1$. This makes $Y = 1/0$, and this value propagates to the output Z. However, now the two inputs of the NAND gate become 0/1 and 1/0, respectively, causing $Y = 1$. Thus, the output Z continues to change as $1/0 \rightarrow 1 \rightarrow 1/0 \rightarrow 1 \ldots$. This means that the fault-free circuit will produce a constant 1 output, while the faulty circuit output will fluctuate between 1 and 0. The period of fluctuation will equal the combined delay of the path including the four gates. The test has two steps: (1) Initialization, $A = 0$; (2) Combinational test, $A = B = 1,\ C = 0$.



(a) Test generation with nine–value logic.



(b) Boolean minimization.



(c) Feedback–free combinational circuit.

Solution of Problem 8.17

We evaluate the fault-free function of the circuit as $Z = ABY + AC$. Further, $Y = \overline{CZ} = \overline{AC}$. Substituting this, we find $Z = AB\overline{C} + AC = A(B + C)$. The function and its two gate combinational (feedback-free) implementation are shown in the figure.

8.18 Simulation-based initialization

The initialization sequence for the circuit of Figure 8.9 (see page 226 of the book) is, $(A, B) = (0,0), (1,0)$. The procedure is illustrated in the following table where the selected vectors are shown in boldface.

Simulation-based initialization of circuit of Figure 8.9								
Phase	Types of vectors	Trial vectors		States			Cost	Remarks
		A	B	$F1$	$F2$	$F3$	func.	
I	Initial condition	X	X	X	X	X	3	Cost=#FFs
	Starting vector	**0**	**0**	0	X	X	2	Cost red.
	Unit Hamm. dist.	**1**	**0**	0	1	1	0	Cost red.
	Circuit initialized (cost=0), Phase I completed.							

The initialization sequence for the circuit of Figure 8.13 (see page 230 of the book) is, (CNT, CLR)= (0,1). The procedure is illustrated in the following table where the selected vector is shown in boldface.

Simulation-based initialization of circuit of Figure 8.13							
Phase	Types of vectors	Trial vectors		States		Cost	Remarks
		CNT	CLR	$FF1$	$FF2$	func.	
I	Initial condition	X	X	X	X	2	Cost=#FFs
	Starting vector	0	0	X	X	2	No cost red.
	Unit Hamm. dist.	1	0	X	X	2	No cost red.
		0	**1**	0	0	0	Cost red.
	Circuit initialized (cost=0), Phase I completed.						

This procedure cannot initialize the circuit in Figure 8.12, because neither $CNT = 0$ nor $CNT = 1$ can force any flip-flop into a defined state. These are the only possible trial vectors. Thus, the initial cost of 2 will never be reduced.

8.19 CONTEST

The *CONTEST* procedure for the s-a-0 fault in Figure 8.3 is as follows:

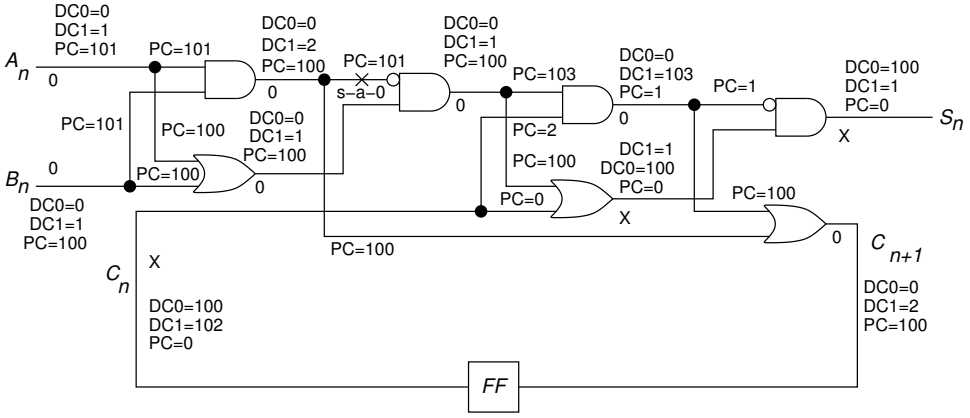
Phase	Type of *vectors*	*Trial vectors* C_n A_n B_n	*Outputs* C_{n+1} S_{n+1}	*Cost function*	*Remarks*
I	Initial condition	X X X	X X	1	Cost = number of uninitialized flip-flops.
	An arbitrary vector	X [0 0]	0 X	0	Phase I completed. Vector 00 accepted.
II	Initialization vector	X 0 0	0 X	∞	Fault simulation of initialization vector; fault not activated.
	Unit Hamming distance vectors	0 0 1	0 1	∞	No cost reduction by any trial vector; vector 10 arbitrarily selected.
		0 [1 0]	0 1	∞	
	Unit Hamming distance vectors	0 0 0	0 0	∞	
		0 [1 1]	1 1/0	0	Fault detected; vector 11 selected.

The test sequence is $(A_n, B_n) = (00), (10), (11)$. The selected vectors are shown in boxes in the table.

8.20 CONTEST

Four steps that lead to the detection of the given s-a-0 faults are illustrated below. For simulation, as a vector is applied to PIs A and B, the next state C_{n+1} value is transferred to the present state input C_n.

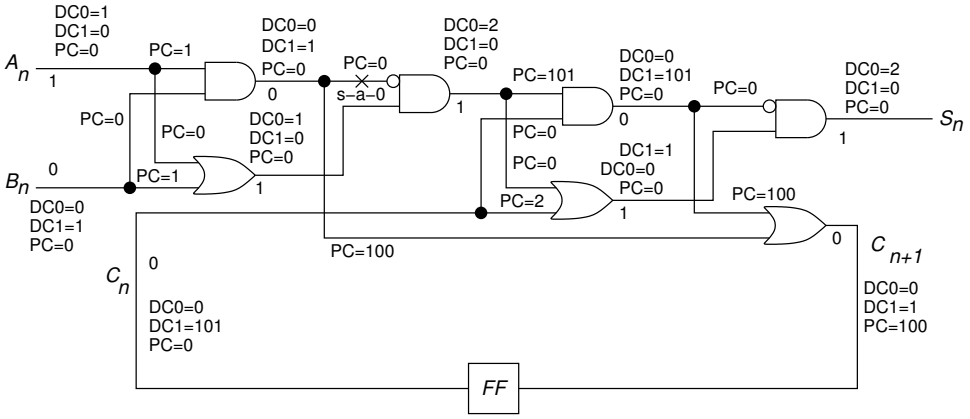
Step 1: Initialization vector 00 is simulated with $C_n = X$. The signal values, dynamic controllabilities, $DC0$ and $DC1$, and propagation cost PC are shown in the following circuit diagram. PC is always 0 for the PO S_n.



Activation cost (AC) equals $DC1$ at the fault site since the fault is of s-a-0 type. We use a weighting factor of 1,000 that multiplies AC. Thus, the fault detection cost for vector 00 is,

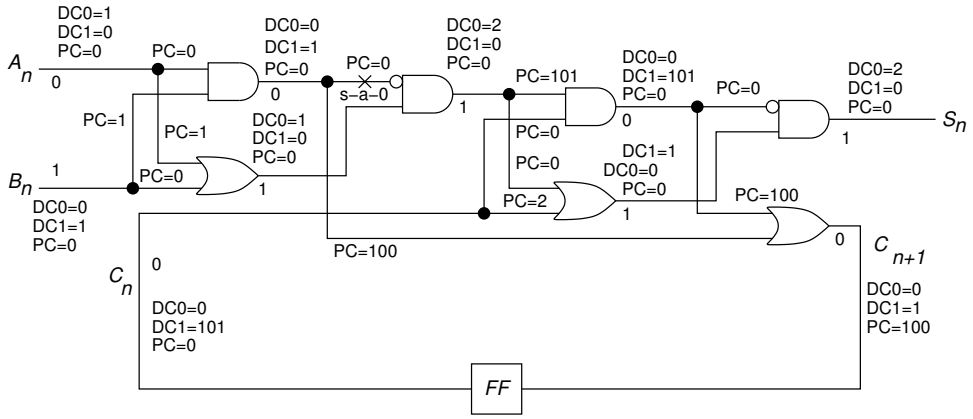
$$\begin{aligned} Cost(00) &= 1000 \times AC + PC \\ &= 1000 \times 2 + 101 = 2101 \end{aligned}$$

Step 2: Unit Hamming distance vector 10 is simulated using the initial state $C_n = 0$ obtained in Step 1. All measures and costs are computed, as shown below,



$$Cost(10) = 1000 \times 1 + 0 = 1000$$

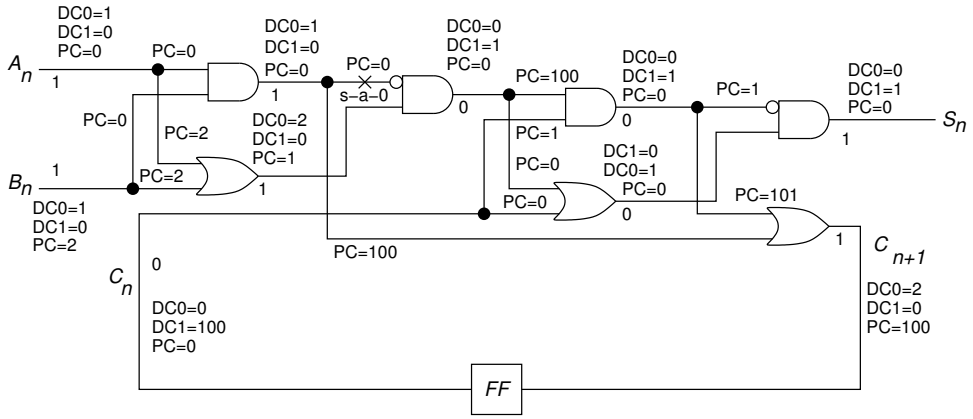
Step 3: Another unit Hamming distance vector 01 is simulated using the initial state of Step 1 and measures and costs are computed, as shown below,



$$Cost(01) = 1000 \times 1 + 0 = 1000$$

Since the cost of both trial vectors is the same, we arbitrarily select the first vector, 10.

Step 4: Now, 10 becomes the current vector. It produces a next state $C_{n+1} = 0$. We try a unit Hamming distance vector 11 as shown below,



$$Cost(11) = 1000 \times 0 + 0 = 0$$

A zero cost indicates that the fault is detected. Thus the complete test sequence is (A_n, B_n) = (0,0), (1,0), (1,1).

Chapter 9: Memory Test

9.1 State coupling faults

In the solution we will use the notation $< i, j >$ to indicate a fault type and (x, y) to indicate the state of the cells i and j respectively. Further, cell i will be the coupled cell (victim cell) and cell j will be coupling cell (aggressor cell).

Fault excitation conditions:

State Coupling Fault $< 1 : 1 >$ is detected if the state is $(0, 0)$ and a 1 is written into cell j.

State Coupling Fault $< 0 : 0 >$ is detected if the state is $(1, 1)$ and a 0 is written into cell j.

State Coupling Fault $< 0 : 1 >$ is detected if the state is $(1, 0)$ and a 1 is written into cell j.

State Coupling Fault $< 1 : 0 >$ is detected if the state is $(0, 1)$ and a 0 is written into cell j.

States of the memory as desired by the excitation conditions:

The state is $(0, 0)$ at the end of march steps M0, M2 and M4.

The state is $(1, 1)$ at the end of march steps M1 and M3.

For $i < j$

The state $(1, 0)$ occurs during the march steps M1 and M4.

The state $(0, 1)$ occurs during the march steps M2 and M3.

For $i > j$

The state $(1, 0)$ occurs during the march steps M2 and M3.

The state $(0, 1)$ occurs during the march steps M1 and M4.

Fault detection:

For $i < j$

The fault $< 1 : 1 >$ is detected during the march step M3.

The fault $< 0 : 0 >$ is detected during the march step M4.

The fault $< 0 : 1 >$ is excited during march step M1 and is detected during M2.

The fault $< 1 : 0 >$ is excited during march step M2 and is detected during M3.

For $i > j$

The fault $< 1 : 1 >$ is detected during the march step M1.

The fault $< 0 : 0 >$ is detected during the march step M2.

The fault $< 0 : 1 >$ is excited during march step M3 and is detected during M4.

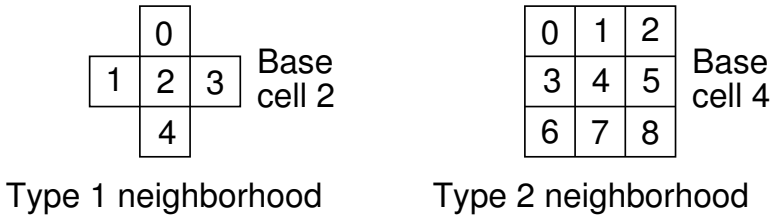
The fault $< 1 : 0 >$ is excited during march step M4 and is detected during M5.

Solution provided by K. K. Saluja

9.2 Address decoder faults

To be proven: A test for a NPSF cannot detect the ADF in which two addresses a and b both access the contents C_b of location b.

Proof by counterexample: Any NPSF test initializes the base cell, then writes the test pattern to the deleted neighborhood cells, and finally reads the base cell to check for a fault. The two neighborhood definitions are shown below: For the



type 1 neighborhood, let cell a be cell 0 and cell b be the base cell 2. For the type 2 neighborhood, let cell a be cell 0 and cell b be the base cell 4.
Then, the active NPSFs

$$\begin{array}{c} 0\ 1\ 3\ 4\ \ 2 \\ <\ \ 0,\downarrow,1,1;1\ \ > \end{array}$$

and

$$\begin{array}{c} 0\ 1\ \ 2\ 3\ 5\ 6\ \ 7\ 8\ 4 \\ <\ \ 0,\downarrow,1,1,1,1,1,1;1\ \ > \end{array}$$

go undetected because any write of the neighborhood cell a instead writes the base cell b. This either removes the fault effect at cell b or prevents sensitization of the fault, since cell a cannot be written. No other ANSPF test will be expected in the sequence to detect this particular fault.

The passive NPSFs, $<\ 1,0,1,1;\uparrow/0\ >$ and $<\ 1,0,1,1,1,1,1,1;\uparrow/0\ >$ will go undetected because any read of the base cell b will produce either an AND of the contents of cells a and b, the OR of a and b, or an intermediate voltage. Since cell a is a 1, the read is apt to produce the good machine value. No other PNPSF test will be expected in the sequence to detect this particular fault.

The static NPSFs $<\ 1,0,1,1;-/0\ >$ and $<\ 1,0,1,1,1,1,1,1;-/0\ >$ both go undetected, because any write of the neighborhood cell a instead writes the base cell b. This either writes the good machine value to the base cell or prevents fault sensitization because cell a cannot be written. No other SNPSF test in the test sequence will be expected to detect this particular fault.

This completes the proof. ■

9.3 Transition faults

The transition fault means that when we set $\overline{CS}$ to a 0, it works. However, if we change $\overline{CS}$ from $0 \rightarrow 1$, it remains at 0. Notice that if the chip powers up with $\overline{CS}$ set to 1, the fault is not active. Conclusions:

1. If the chip powers up with $\overline{CS} = 0$, then it is permanently selected.
2. Otherwise, the first time we select the chip, it works, but it remains permanently selected.

This will appear to be an address decoder fault.

9.4 Port arbitration faults

An arbitration logic test is as follows:

1. Write a 0 to location x through the DRAM port.
2. Simultaneously write a 0 to x through the DRAM port and write a 1 to x through the SRAM port.
3. Read $\overline{BUSY}$ for both the SRAM and DRAM ports. If both $\overline{BUSY}$ lines are 1, then the chip is faulty. If only one line is 1, then the chip is good.

9.5 ROM testing

The CRC based ROM test requires readout of all n memory locations. If it is 1 bit per location, an LFSR compresses the response. If there are B bits per word, a B bit MISR compresses the response. In either case, the LFSR or MISR must be initialized to 0s. There is one extra read from the memory to fetch the CRC stored in the ROM, which must be compared with the MISR contents.

For a 1 bit per word memory, where K is the number of bits in the CRC, this leads to $n + K$ reads, which is $O(n)$.

For a B bits per word memory, this leads to $\frac{n}{B} + 1$ reads (assuming that $B = K$), which is $O(n)$.

9.6 Graphs

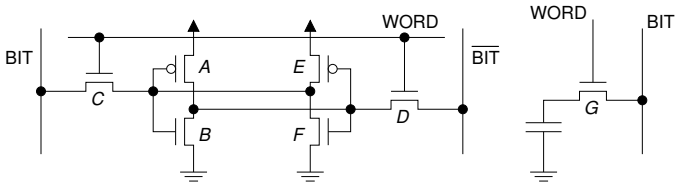
A Hamiltonian graph traversal visits each node in the graph exactly once, while an Eulerian traversal traverses each edge exactly once.

9.7 Stuck-open faults

Let us denote the components of the given IFA-13 march test algorithm by $M0$ through $M8$:

$$\{\ M0: \ \Uparrow(w0);\ M1: \ \Uparrow(r0,w1,r1);\ M2: \ \Uparrow(r1,w0,r0);$$
$$M3: \ \Downarrow(r0,w1,r1);\ M4: \ \Downarrow(r1,w0,r0);\ M5: \ Delay;$$
$$M6: \ \Uparrow(r0,w1);\ M7: \ Delay;\ M8: \ \Uparrow(r1)\ \}$$

We also denote the transistors in the memory cells as follows:



To prove that IFA-13 detects all stuck-open faults in the memory, we proceed as follows:

I. For transistor A stuck-open (sop) the necessary test conditions are:

(i) Write a 0 (M0)
(ii) Write a 1 (M1)
(iii) Read a 1 (M1)

II. For transistor B sop, the necessary test conditions are:

(i) Write a 1 (M1)
(ii) Write a 0 (M2)
(iii) Read a 0 (M2)

III. For transistor C sop, the necessary test conditions are:

(i) Write a 0 (M0)
(ii) Write a 1 (M1)
(iii) Read a 1 (M1)

IV. For transistor D sop, the necessary test conditions are:

(i) Write a 1 (M1)
(ii) Write a 0 (M2)
(iii) Read a 0 (M2)

V. For transistor E sop, the necessary test conditions are:

(i) Write a 1 (M1)
(ii) Write a 0 (M2)
(iii) Read a 0 (M2)

VI. For transistor F sop, the necessary test conditions are:

(i) Write a 0 (M0)
(ii) Write a 1 (M1)
(iii) Read a 1 (M1)

VII. For transistor G sop, the necessary test conditions are:

(i) Write a 0 (M0)
(ii) Read a 0 (M1)

That completes the proof. ■

9.8 Test types

The *probe test* can simply write and read a few memory locations to verify that the chip does not have major damage. It need not test very many cells. It is done separately from functional test because it needs a short test sequence to keep the test cheap. Also, it needs a flying-probe tester, since the chip is not packaged. That is another reason why probe test is done separately.

The *contact test* forces a current out of a pin and then precisely measures the pin voltage, which may be negative. It requires an analog tester with a parametric measurement unit (PMU), whereas the functional tests only require a digital tester. That is why the contact test is not combined with functional test.

9.9 Idempotent coupling faults

Necessary condition for idempotent coupling fault test: For all coupled cells, each should be read after a series of possible CFids may have happened, such that the sensitized CFids do not mask each other (the coupled cells are read while their state is opposite from the good machine state.)

We consider the MARCH C− test:

$$\{\ M0: \ \Updownarrow (w0); M1: \ \Uparrow (r0, w1); M2: \ \Uparrow (r1, w0);$$
$$M3: \ \Downarrow (r0, w1); M4: \ \Downarrow (r1, w0); M5: \ \Updownarrow (r0)\ \}$$

For a coupling cell C_j and a coupled cell C_i, we have

Theorem: MARCH C− detects all CFid faults, $<\uparrow;0>$, $<\downarrow;1>$, $<\uparrow;1>$, $<\downarrow;0>$.

Proof:

I. For the fault $<\uparrow;0>$ the necessary test conditions are:

- **(a)** First write a 0 to j and a 1 to i
- **(b)** Second, write a 1 to j
- **(c)** Read cell i and check for a 1 before changing i or j

Note that,

1. If $Addr(j) < Addr(i)$, $M2$ and $M3$ satisfy (a), $M3$ satisfies (b), and $M4$ satisfies (c).
2. If $Addr(j) > Addr(i)$, $M0$ and $M1$ satisfy (a), $M1$ satisfies (b), and $M2$ satisfies (c).

II. For the fault $<\downarrow, 0>$ the necessary test conditions are:

- **(a)** Write a 1 to i and j

(b) Write a 0 to j

(c) Check i for a 1 before changing i or j

Note that,

1. If $Addr(j) < Addr(i)$, $M2$ satisfies (b) and (c).
2. If $Addr(j) > Addr(i)$, $M3$ satisfies (a), and $M4$ satisfies (b) and (c).

III. for the fault $<\uparrow;1>$ the necessary conditions are:

(a) Write a 0 to i and j

(b) Write a 1 to j

(c) Check i for a 0 before changing i or j

Note that,

1. If $Addr(j) < Addr(i)$, $M0$ satisfies (a), and $M1$ satisfies (b) and (c).
2. If $Addr(j) > Addr(i)$, $M2$ satisfies (a), and $M3$ satisfies (b) and (c).

IV. For the fault $<\downarrow;1>$ the necessary conditions are:

(a) Write a 0 to i and a 1 to j

(b) Write a 0 to j

(c) Check i for a 0 before changing i or j

Note that,

1. If $Addr(j) < Addr(i)$, $M3$ and $M4$ satisfy (a), $M4$ satisfies (b), and $M5$ satisfies (c).
2. If $Addr(j) > Addr(i)$, $M1$ and $M2$ satisfy (a), $M2$ satisfies (b), and $M3$ satisfies (c).

That completes the proof. ■

For an n bit memory, the complexity of MARCH C− is $O(10n)$.

9.10 Fault modeling

(a) A *state coupling fault* (SCF) $< i, j >$ is a memory fault where the coupling cell i entering the state 0 or 1 causes the coupled cell j to enter the state 0 or 1. These are denoted as $< 0;0 >$, $< 0;1 >$, $< 1;0 >$ and $< 1;1 >$.

(b) An *inversion coupling fault* (CFin) $< i, j >$ is where the coupling cell i having a transition causes the coupled cell j to invert its state. These are denoted as $<\uparrow;\updownarrow>$, $<\downarrow;\updownarrow>$.

(c) An *idempotent coupling fault* (CFid) $< i, j >$ is where the coupling cell i having a transition causes the coupled cell j to enter a particular state. These are denoted as $<\uparrow; 0>$, $<\uparrow; 1>$, $<\downarrow; 0>$, and $<\downarrow; 1>$.

(d) A *dynamic coupling fault* (CFdyn) is where a read or write of a cell in one word forces the contents of a cell in another word to 0 or 1. These are denoted as $< r0|w0; 0 >$, $< r0|w0; 1 >$, $< r1|w1; 0 >$, and $< r1|w1; 1 >$.

(e) A rising (falling) *transition fault* (TF) in a memory cell that can come up in either the 0 or 1 state, but any attempt to change its state from 0 to 1 (1 to 0) fails. These are denotes as $<\uparrow; 0>$ and $<\downarrow; 1>$.

(f) An *active neighborhood pattern sensitive fault* (ANSPF) causes the base cell to change due to a pattern and transition in the deleted neighborhood. The base cell can go to 0, 1, or invert.

(g) A *passive neighborhood pattern sensitive fault* (PNPSF) prevents the base cell from changing when a particular pattern exists in the deleted neighborhood.

(h) A *static neighborhood pattern sensitive fault* (SNPSF) forces the base cell into a particular state when a particular pattern exists in the deleted neighborhood.

(i) A *data retention fault* causes a memory cell to forget its content over time, usually due to a damaged SRAM pullup device or a damaged DRAM capacitor.

(j) An *address decoder fault* in a memory causes, 1) an address i to instead access location j, 2) an address i to access no location, or 3) address i to simultaneously access multiple locations.

9.11 Memory test algorithms

We rigorously prove that the MARCH C− test detects all inversion coupling faults (CFin).

The MARCH C− test is,

$$\{\ M0: \Updownarrow (w0); M1: \Uparrow (r0, w1); M2: \Uparrow (r1, w0);$$
$$M3: \Downarrow (r0, w1); M4: \Downarrow (r1, w0); M5: \Updownarrow (r0)\ \}$$

and the inversion coupling faults are $<\uparrow; \updownarrow>$ and $<\downarrow; \updownarrow>$.

Necessary condition: For all cells that are coupled, each should be read after series of possible CFins may have occurred, and the number of coupled cell transitions must be odd.

Fault $<\uparrow; \updownarrow>$**:** *Address of coupled cell* i $>$ *address of coupling cell* j. Cell j initialized to 0 by $M0$, j is made to $\uparrow$ by $M1$, coupled cell i set to 0 by $M0$, unexpected inversion detected by $M1$, number of coupled cell inversions = 1. *Address of coupled cell* i $<$ *address of coupling cell* j. Cell j initialized to 0 by $M2$, j made to $\uparrow$ by $M3$, coupled cell i set to 0 by $M2$, unexpected inversion detected by $M3$, number of coupled cell inversions = 1.

Fault $<\downarrow;\updownarrow>$**:** *Address of coupled cell* i $>$ *address of coupling cell* j. Cell j initialized to 1 by $M1$, j is made to $\downarrow$ by $M2$, coupled cell i set to 1 by $M1$, unexpected inversion detected by $M2$, number of coupled cell inversions = 1. *Address of coupled cell* i $<$ *address of coupling cell* j. Cell j initialized to 1 by $M3$, j is made to $\downarrow$ by $M4$, coupled cell i set to 1 by $M3$, unexpected inversion detected by $M4$, number of coupled cell inversions = 1.

That completes the proof. ■

9.12 Stuck-at faults

We rigorously prove that the MATS++ test catches all stuck-at faults.
The MATS++ test is,

$$\{\ M0:\ \Updownarrow(w0);\ M1:\ \Uparrow(r0,w1);\ M2:\ \Downarrow(r1,w0,r0)\ \}$$

and the stuck-at faults are $<\forall/0>$ and $<\forall/1>$.

Necessary condition: For each cell, a 0 and a 1 must be read.

Fault $<\forall/0>$**:** S-a-0 fault is sensitized by writing a 1 to the cell in $M1$. S-a-0 fault is detected by $M2$ when a 0 is read from the cell, while a 1 was expected.

Fault $<\forall/1>$**:** S-a-1 fault is sensitized by writing a 0 to the cell in $M0$. S-a-1 fault is detected by $M1$ when a 1 is read from the cell, while a 0 was expected.

That completes the proof. ■

9.13 Dynamic coupling faults

We rigorously prove that the MARCH C− test detects all dynamic coupling faults. MARCH C− test is,

$$\{\ M0:\ \Updownarrow(w0);\ M1:\ \Uparrow(r0,w1);\ M2:\ \Uparrow(r1,w0);$$
$$M3:\ \Downarrow(r0,w1);\ M4:\ \Downarrow(r1,w0);\ M5:\ \Updownarrow(r0)\ \}$$

and dynamic coupling faults are $<r0|w0;0>$, $<r0|w0;1>$, $<r1|w1;0>$ and $<r1|w1;1>$.

Necessary condition: After initializing the coupled cell, a read (write) of the coupling cell must be followed by a read of the coupled cell, without any intervening operations on the coupled cell.

Fault $<r0|w0;0>$**:** *Address of coupled cell* i $>$ *Address of coupling cell* j.
For a write,
i initialized by M1, j written by M2,
i checked by M2 (fault detected).
For a read,

i initialized by M3, j read by M3,
i checked by M4 (fault detected).
Address of coupled cell i $<$ *Address of coupling cell* j.
For a write,
i initialized by M3, j written by M4,
i checked by M4 (fault detected).
For a read,
i initialized by M3, j read by M4,
i checked by M4 (fault detected).

Fault $< r0|w0;1 >$**:** *Address of coupled cell* i $>$ *Address of coupling cell* j.
For a write,
i initialized by M4, j written by M4,
i checked by M5 (fault detected).
For a read,
i initialized by M0, j read by M1,
i checked by M1 (fault detected).
Address of coupled cell i $<$ *Address of coupling cell* j.
For a write,
i initialized by M2, j written by M2
i checked by M3 (fault detected).
For a read,
i initialized by M2, j read by M3,
i checked by M3 (fault detected).

Fault $< r1|w1;0 >$**:** *Address of coupled cell* i $>$ *Address of coupling cell* j.
For a write,
i initialized by M1, j written by M1,
i checked by M2 (fault detected).
For a read,
i initialized by M1, j read by M2,
i checked by M2 (fault detected).
Address of coupled cell i $<$ *Address of coupling cell* j.
For a write,
i initialized by M3, j written by M3
i checked by M4 (fault detected).
For a read,
i initialized by M4, j read by M4,
i checked by M4 (fault detected).

Fault $< r1|w1;1 >$**:** *Address of coupled cell* i $>$ *Address of coupling cell* j.
For a write,
i initialized by M0, j written by M1,
i checked by M1 (fault detected).
For a read,
i initialized by M0, j read by M2,

i checked by M2 (fault detected).
$Address\ of\ coupled\ cell\ i\ <\ Address\ of\ coupling\ cell\ j$.
For a write,
i initialized by M2, j written by M3
i checked by M3 (fault detected).
For a read,
i initialized by M2, j read by M2,
i checked by M3 (fault detected).

That completes the proof. ■

9.14 Data retention faults

We prove that IFA-13 catches all data retention faults. IFA-13 is,

$$\{\ M0:\ \Uparrow(w0);\ M1:\ \Uparrow(r0,w1,r1);\ M2:\ \Uparrow(r1,w0,r0);$$
$$M3:\ \Downarrow(r0,w1,r1);\ M4:\ \Downarrow(r1,w0,r0);\ M5:\ Delay;$$
$$M6:\ \Uparrow(r0,w1);\ M7:\ Delay;\ M8:\ \Uparrow(r1)\ \}$$

and the faults are $< 1/0\ after\ time\ delay >$ and $< 0/1\ after\ time\ delay >$.

***Necessary condition*:** Each cell must have a 0(1) written to it, and after a suitable delay (e.g., 100ms), a 0(1) must be read back from the cell.

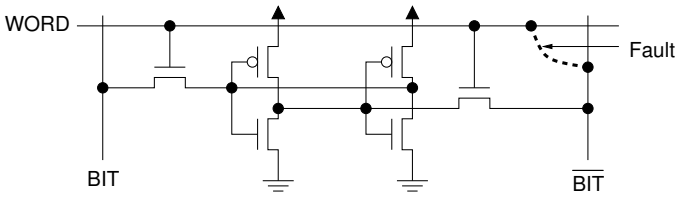
Fault $< 1/0\ after\ time\ delay >$**:** M6 sensitizes the fault by writing a 1, M7 provides the necessary time delay, and M8 detects the fault when a 0 is read but a 1 was expected.

Fault $< 0/1\ after\ time\ delay >$**:** M4 sensitizes the fault by writing a 0, M5 provides the necessary time delay, and M6 detects the fault when a 1 is read but a 0 was expected.

That completes the proof. ■

9.15 SRAM physical faults

A physical fault shorting the $\overline{BIT}$ line to the $WORD$ line in a SRAM cell is shown in the figure below. The fault models are discussed next.



1. When $WORD = 0$ (driven), $\overline{BIT}$ is always forced to be 0 for all reads/writes in this column. This causes all writes/reads in the column to be a 1.

2. When $WORD = 1$ (driven), $\overline{BIT}$ is always forced to be 1 for all reads/writes for this cell. If we are writing 0 into this cell, no error occurs.

 (i) When writing 1 into this cell, a 0 will instead be written only if the $WORD$ line driver is stronger than the BIT line driver. Otherwise, no error occurs.

 (ii) When reading a 0 from this cell no error occurs. When reading a 1 from this cell, an error occurs only if the $WORD$ line driver is stronger than the BIT line driver.

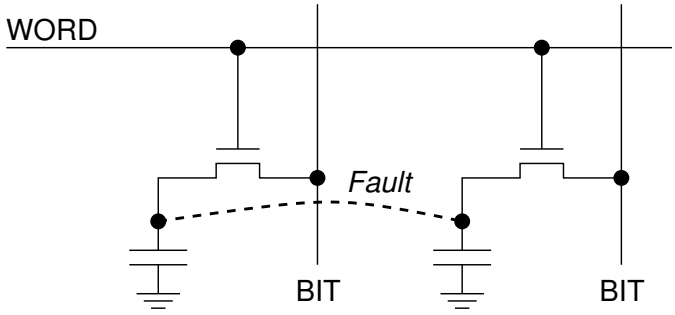
 Case 1 is state coupling fault $< 0; 1 >$ *between the faulty crosspoint cell and all cells in the same column.*

 We assume a $WORD$ *line driver stronger than a* $\overline{BIT}$ *line driver. Then cases 2(i) and 2(ii) are a SA0 fault in the crosspoint cell.*

3. Note that whenever $\overline{BIT}$ is charged for this column, that it also activates $WORD$ for the row containing the faulty cell. This makes all cells in the row having the faulty cell active. The result depends on the column address decoder and the $BIT/\overline{BIT}$ driver. If drivers other than those of the faulty column are also activated (which is usually true with a word-oriented SRAM), then any write of a 0 into any part of the affected column also activates a write into the faulty row, at least for the rest of the bits in this memory word. This would be a state coupling fault $< 0; 0 >$ or $< 1; 1 >$ between the bits in the row intended to be addressed and the corresponding bits in the row with the crosspoint fault.

9.16 DRAM physical faults

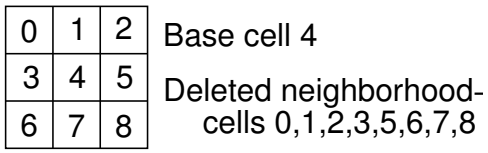
Consider the fault, two DRAM capacitors shorted together, as shown in the figure below.



This is a state coupling pair of faults, $< 0; 0 >$ and $< 1; 1 >$, between the two cells.

9.17 Neighborhood PSFs

The two group method cannot be used with the type-2 neighborhood for a pattern sensitive fault (PSF) test.



Type–2 neighborhood

Duality is the property that all tests for cell 4 as a base cell also provide the necessary test patterns when cells 1, 3, 5 or 7 are considered to be the base cell. This fails for the type-2 neighborhood, bacause the test patterns for cell 4 as the base cell do not provide all necessary test patterns when diagonal cells 0, 2, 6 or 8 are considered to be the base cell.

9.18 Data retention faults

A data retention fault occurs in a DRAM when the side of memory storage capacitor connected to the word line transistor has a significant charge leakage fault either to V_{SS} or to V_{DD}.

9.19 Write recovery faults

A write recovery fault occurs when a write is immediately followed by a read/write at a different address. It is caused by a fault in the sense amplifier that causes it to saturate its transistors after the first write. The immediately following read or write will fail if the data value is the opposite of the data value for the first write. This happens because the fault prevents the sense amplifier transistors from leaving saturation and applying the opposite data value.

9.20 Bridging faults

First, we prove that a MARCH test for a CFid will also detect the AND and OR bridging faults.

I. Necessary steps for CFid $<\uparrow; 0>$:

(a) Write 0 to coupling cell j and 1 to coupled cell i

(b) Write 1 to j

(c) Read cell i and check for a 1 before changing i or j

Note that:

1. If $Addr(j) < Addr(i)$
 $M0: \Updownarrow(w0)$; $M1: \Downarrow(w1)$ satisfies (a) and (b), and an immediate
 $M2: \Updownarrow(r1)$ satisfies (c).
2. If $Addr(j) > Addr(i)$
 $M3: \Updownarrow(w0)$; $M4: \Uparrow(w1)$ satisfies (a) and (b), and an immediate
 $M5: \Updownarrow(r1)$ satisfies (c).

II. Necessary steps for CFid $<\downarrow;0>$:

(a) Write a 1 to i and j

(b) Write a 0 to j

(c) Check i for a 1 before changing i or j

Note that:

1. If $Addr(j) < Addr(i)$
 $M6: \Updownarrow(w1)$ satisfies (a), $M7: \Uparrow(r1,w0)$ satisfies (b) and (c).
2. If $Addr(j) > Addr(i)$
 $M8: \Updownarrow(w1)$ satisfies (a), $M9: \Downarrow(r1,w0)$ satisfies (b) and (c).

III. Necessary steps for CFid $<\uparrow;1>$:

(a) Write a 0 to j and i

(b) Write a 1 to j

(c) Check i for a 0 before changing i or j

Note that:

1. If $Addr(j) < Addr(i)$
 $M10: \Updownarrow(w0)$ satisfies (a), $M11: \Uparrow(r0,w1)$ satisfies (b) and (c).
2. If $Addr(j) > Addr(i)$
 $M12: \Updownarrow(w0)$ satisfies (a), $M13: \Downarrow(r0,w1)$ satisfies (b) and (c).

IV. Necessary steps for CFid $<\downarrow;1>$:

(a) Write a 1 to j and 0 to i

(b) Write a 0 to j

(c) Check i for a 0 before changing i or j

Note that:

1. If $Addr(j) < Addr(i)$
 $M14: \Updownarrow(w1)$; $M15: \Downarrow(w0)$ satisfies (a) and (b) and an immediate
 $M16: \Updownarrow(r0)$ satisfies (c).
2. If $Addr(j) > Addr(i)$
 $M17: \Updownarrow(w1)$; $M18: \Uparrow(w0)$ satisfies (a) and (b) and an immediate
 $M19: \Updownarrow(r0)$ satisfies (c).

That completes the proof of the first part. ■

Next, to prove that CFid MARCH test also detects all ABFs: $< 0, 0/0, 0 >$, $< 0, 1/0, 0 >$, $< 1, 0/0, 0 >$, $< 1, 1/1, 1 >$, and OBFs: $< 0, 0/0, 0 >$, $< 0, 1/1, 1 >$, $< 1, 0/1, 1 >$, $< 1, 1/1, 1 >$, we need only check for the middle two ABFs and OBFs.

I. Necessary steps for ABF $< 0, 1/0, 0 >$, where bit order is (j, i), are,

- **(a)** Write 0 to j and 1 to i.
- **(b)** Read i and report error if it became 0. Sequence $M0 - M2$ in I(1) of the previous proof does this if $Addr(j) < Addr(i)$, otherwise, sequence $M3 - M5$ in I(2) does this.

II. Necessary steps for ABF $< 1, 0/0, 0 >$ (bit order j, i) are,

- **(a)** Write 1 to j and 0 to i.
- **(b)** Read j and report error if it became 0. Sequence $M0 - M2$ in I(1) of the previous proof does this for $Addr(j) > Addr(i)$, otherwise, sequence $M3 - M5$ does this if $Addr(j) < Addr(i)$.

III. Necessary steps for OBF $< 1, 0/1, 1 >$ (bit order j, i) are,

- **(a)** Write 1 to j and 0 to i.
- **(b)** Read i and report error if it became 1. Sequence $M14 - M16$ in IV(1) of the previous proof does this for $Addr(j) < Addr(i)$, otherwise, sequence $M17 - M19$ in IV(2) does this if $Addr(j) > Addr(i)$.

IV. Necessary steps for OBF $< 0, 1/1, 1 >$ (bit order j, i) are,

- **(a)** Write a 0 to j and 1 to i.
- **(b)** Read j and report error if it became 1. Sequence $M14 - M16$ in IV(1) of the previous proof does this if $Addr(j) > Addr(i)$, otherwise, sequence $M17 - M19$ in IV(2) does this if $Addr(j) < Addr(i)$.

That completes the proof of the second part. ■

An alternative, and much simpler, proof:

ABF $< 0, 1/0, 0 >$ and $< 1, 0/0, 0 >$ are equivalent to bidirectional coupling fault $<\uparrow; 0 >$.

OBF $< 0, 1/1, 1 >$ and $< 1, 0/1, 1 >$ are equivalent to bidirectional coupling fault $<\downarrow; 1 >$ ■

9.21 State coupling faults

We prove that a MARCH test for CFid will also detect state coupling faults.

State coupling fault test for $< 0; 0 >$ is covered by CFid test $<\uparrow; 0 >$, since the step writing 1 to the coupling cell is not needed.

SCF test for $< 0; 1 >$ is covered by CFid test $<\uparrow; 1 >$, since the step writing 1 to the coupling cell is not needed.

SCF test for $< 1; 0 >$ is covered by CFid test $<\downarrow; 0 >$, since the step writing 0 to the coupling cell is not needed.

SCF test for $< 1; 1 >$ is covered by CFid test $<\downarrow; 1 >$, since the step writing 0 to the coupling cell is not needed.

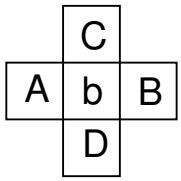
That completes the proof. ■

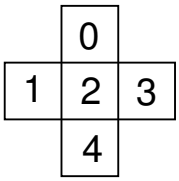
9.22 Neighborhood PSFs

We write the steps (in pseudo-code) for a test to detect the passive neighborhood pattern sensitive faults (PNPSF),

$$< 1, 0, 1, 0; \uparrow /0 > \text{ and } < 0, 1, 0, 1; \downarrow /1 >$$

using the two-group method and type-1 neighborhood. The test need not be the optimal one.





Write '0' to base cell;
Write "1010" to cells 0, 1, 3, 4;
Write '1' to base cell;
Read base cell (test fails in it is '0');
Write '1' to base cell;
Write "0101" to cells 0, 1, 3, 4; Write '0' to base cell;
Read base cell (test fails in it is '1');

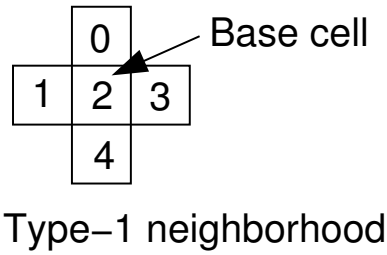
9.23 Neighborhood PSFs

We write the steps (in pseudo-code) for a PNPSF test to detect the faults,

$$< 0, 0, 0, 0; \uparrow /0 > \text{ and } < 1, 1, 1, 1; \downarrow /1 >$$

using the two-group method and type-1 neighborhoods (see figure below):

Write '0' to base cell;
Write "0000" to cells 0, 1, 3, 4;
Write '1' to base cell;
Read the base cell (test fails if it is '0');



Type–1 neighborhood

Write ‘1’ to base cell;
Write “1111” to cells 0, 1, 3, 4;
Write ‘0’ to base cell;
Read the base cell (test fails if it is ‘1’);

Chapter 10: DSP-Based Analog and Mixed-Signal Test

10.1 Unit test period

$$\frac{F_t}{F_s} = \frac{M}{N} = \frac{2,010Hz}{8,000Hz} = \frac{201}{800}$$

That is, $M = 201$ and $N = 800$. Unit test period is obtained as,

$$\begin{aligned} \text{UTP} &= \frac{M}{F_t} = \frac{201}{2,010Hz} = 0.1\ sec \\ \text{Primitive frequency, } \Delta &= \frac{1}{UTP} = 10Hz \end{aligned}$$

10.2 Unit test period

$$\begin{aligned} \text{Primitive frequency, } \Delta &= 20Hz \\ \text{Unit test period, } UTP &= \frac{1}{\Delta} = 0.05\ sec \\ M &= \frac{F_t}{\Delta} = \frac{2020Hz}{20Hz} = 101 \end{aligned}$$

Would like $N = 600$, which is relatively prime to 101, so it is OK.

$$F_s = N \times \Delta = 600 \times 20Hz = 12,000Hz$$

10.3 Unit test period

$$\begin{aligned} \frac{F_t}{F_s} &= \frac{M}{N} = \frac{2,000Hz}{16,000Hz} = \frac{1}{8} \\ \Delta &= \frac{1}{UTP} = \frac{1}{50ms} = 20Hz \end{aligned}$$

but M and N are not relatively prime. We get only eight unique samples and as the

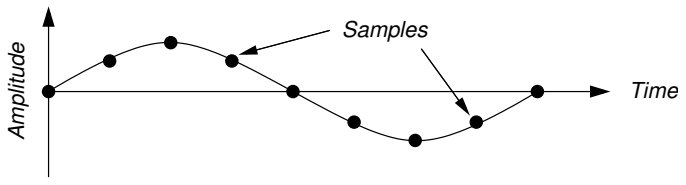


figure shows every ninth sample repeats. *This is a totally inadequate sample set.*

10.4 Unit test period

$$\begin{aligned} UTP &= 40\ msec \\ \Delta &= \frac{1}{UTP} = 25Hz \\ N &= \frac{F_s}{\Delta} = \frac{8,000Hz}{25Hz} = 320 \\ M &= \frac{F_t}{\Delta} = \frac{400Hz}{25Hz} = 16 \\ \frac{M}{N} &= \frac{16}{320} = \frac{1}{20};\ M \text{ and } N \text{ are not relative primes!} \end{aligned}$$

If we choose $M = 15$, M and N are still not relative primes.
$\boxed{\text{Choose } M = 17,}$ $\frac{M}{N} = \frac{17}{320}$; M and N are relative primes. So we get 320 unique samples.

$$F_t = M \times \Delta = 17 \times 25Hz = \boxed{425\text{Hz}}$$

10.5 Unit test period

$$\begin{aligned} \Delta &= \frac{1}{p} = \frac{1}{40\ msec} = 25Hz \\ N &= \frac{F_s}{\Delta} = \frac{8,000\ s/s}{25Hz} = 320 \end{aligned}$$

We must change Δ and p to get $N = 400$.

$$\Delta = \frac{8,000\ s/s}{400} = 20Hz$$

If $F_t = 2,000Hz$, $M = \frac{F_t}{\Delta} = \frac{2,000Hz}{20Hz} = 100$, and $\frac{M}{N} = \frac{100}{400} = \frac{1}{4}$, which gives only 4 samples. So, choose either $M = 99$ or $M = 101$.

For $M = 99$, $F_t = M \times \Delta = 99 \times 20Hz = 1,980Hz$, and

For $M = 101$, $F_t = M \times \Delta = 101 \times 20Hz = 2,020Hz$.

In both cases, we get 400 unique samples.

10.6 Correlation

$$\begin{aligned} RMS(A) &= 2.8214 \\ RMS(B) &= 5.6709 \\ G &= \frac{1}{RMS(A) \cdot RMS(B) \cdot 1} \\ &= 0.6525 \end{aligned}$$

$$\text{For } \tau = 0,\ R = G \cdot \int^{P} A(2\pi t) \cdot B(2\pi t) dt$$
$$R = 0, \text{ due to Fourier's second principle.}$$

10.7 Correlation

The correlation of A and B is computed by the following MATLAB program:

```
a = (1/30);
i = 1;
for t = 0:1:(1/3)
A = 16 * sin (6 * pi * t);
B = 14 * sin ((6 * pi * t) + 10);
C = A * B;
end
C = sum(C);
Arms = 16/sqrt(2);
Brms = 14/sqrt(2);
UTP = 1/3;
K = (Arms * Brms * UTP);
G = 1/K;
Corr = G * C

Corr =

   7.9948e-16
```

$$\boxed{\text{Correlation} = 7.9948 \times 10^{-16}}$$

10.8 Multi-tone testing

(a) First harmonics or fundamentals: $6f,\ 19f,\ 27f$

(b) Second harmonics: $12f,\ 38f,\ 54f$

(c) Third harmonics: $18f,\ 57f,\ 81f$

(d) Fourth harmonics: $24f,\ 76f,\ 108f$

(e) First-order intermodulation: $6f,\ 19f,\ 27f$
Second-order intermodulation: $25f,\ 33f,\ 46f,\ 13f,\ 21f,\ 8f$

(f) Third-order intermodulation products (sum or difference between any fundamental and a second-order intermodulation product):

$19f,\ 27f,\ 40f,\ 7f,\ 15f,\ 2f$
$31f,\ 39f,\ 52f,\ 19f,\ 27f,\ 14f$
$6f,\ 14f,\ 27f,\ 6f,\ 2f,\ 11f$

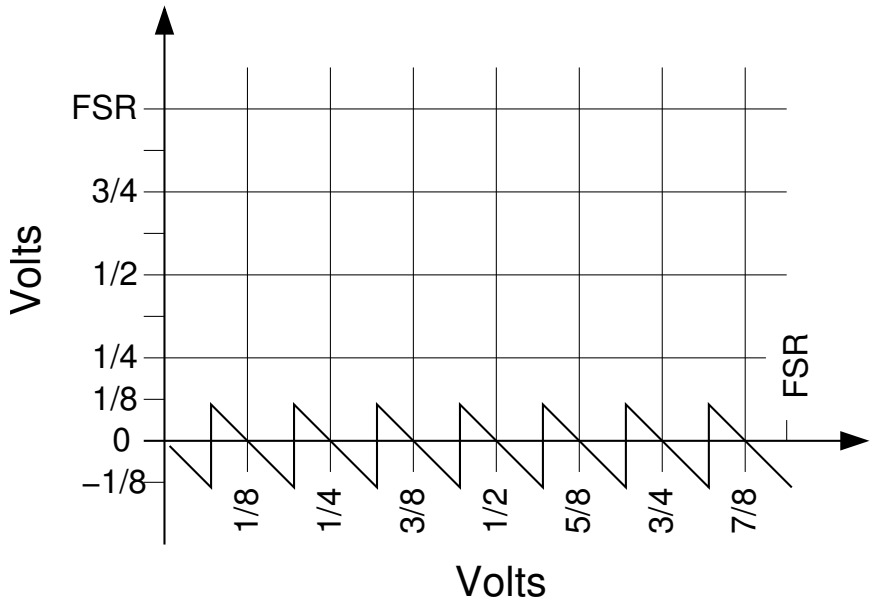
$44f,\ 52f,\ 65f,\ 32f,\ 40f,\ 27f$
$2f,\ 6f,\ 19f,\ 14f,\ 6f,\ 19f$
$52f,\ 60f,\ 73f,\ 40f,\ 48f,\ 35f$

10.9 CODEC testing

$$\begin{aligned}
THD &= \frac{\sqrt{10^{H_2/10} + 10^{H_3/10} + \cdots}}{10^{H_1/20}} \\
H_2(dB) &= 10 \log \frac{0.5mW}{2mW} = -6.020599913dB \\
H_3(dB) &= 10 \log \frac{0.2mW}{2mW} = -10dB \\
H_1 dB) &= 0 \\
THD &= \sqrt{10^{-0.6020599913} + 10^{-1}} \\
&= 0.591608 \\
&= 59.1608\%
\end{aligned}$$

10.10 ADC quantization error

We define this as: *Actual digitized waveform − Original waveform*, to be consistent with the definitions of DNL and INL. However, the negative of this definition may also be considered to be correct.



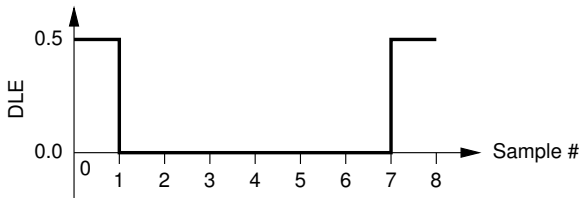
10.11 ADC DLE and ILE

For the ADC of Figure 10.8(a) (page 324 of the book), for parts (a)-(d) of the problem, we assume 32 samples. We add 2 samples on either side for virtual codes.

Code tally	T(0) 6	T(1) 4	T(2) 4	T(3) 4	T(4) 4	T(5) 4	T(6) 4	T(7) 6
DLE	D(0) 0.5	D(1) 0	D(2) 0	D(3) 0	D(4) 0	D(5) 0	D(6) 0	D(7) 0.5
DNL (RMS LSB)	0.5							
DNL (Worst)	0.5							
Transfer char.	C(0) 0	C(1) 5	C(2) 9	C(3) 13	C(4) 17	C(5) 21	C(6) 25	C(7) 30
ILE	E(0) 0	E(1) 0.25	E(2) 0.25	E(3) 0.25	E(4) 0.25	E(5) 0.25	E(6) 0.25	E(7) 0.5
INL (RMS LSB)	0.2795							
INL (Worst)	0.5							

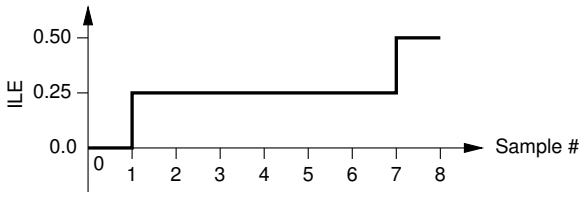
Average step size = 4

(a) Graph of DLE function:



(b) DNL is the worst case DLE value (here it is 0.5 LSB.)

(c) Graph of ILE function:



(d) The INL is the worst case ILE value (here it is 0.5 LSB.)

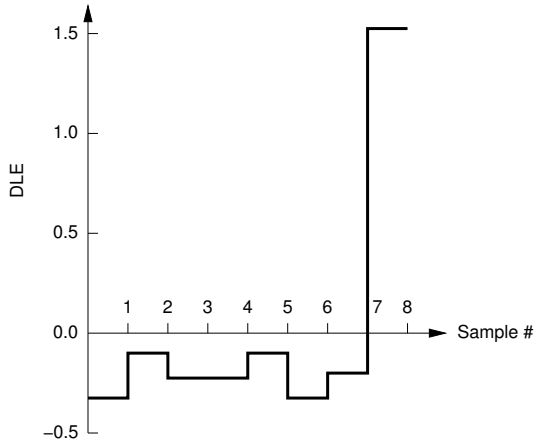
10.12 ADC DLE and ILE

For the ADC of Figure 10.9(a) (page 324 of the book), for parts (a)-(d) of the problem, we assume 64 samples. We add 4 samples on either side for virtual codes.

Code tally	T(0)	T(1)	T(2)	T(3)	T(4)	T(5)	T(6)	T(7)
	6	8	7	7	8	6	7	23
DLE	D(0)	D(1)	D(2)	D(3)	D(4)	D(5)	D(6)	D(7)
	−0.33	−0.11	−0.22	−0.22	−0.11	−0.33	−0.22	1.55
DNL (RMS LSB)	0.5905							
DNL (Worst)	1.55							
Transfer char.	C(0)	C(1)	C(2)	C(3)	C(4)	C(5)	C(6)	C(7)
	0	7	14.5	21.5	29.0	36.0	42.5	57.5
ILE	E(0)	E(1)	E(2)	E(3)	E(4)	E(5)	E(6)	E(7)
	0	−0.22	−0.385	−0.605	−0.77	−0.99	−1.265	−0.6
INL (RMS LSB)	0.71553							
INL (Worst)	−1.265							

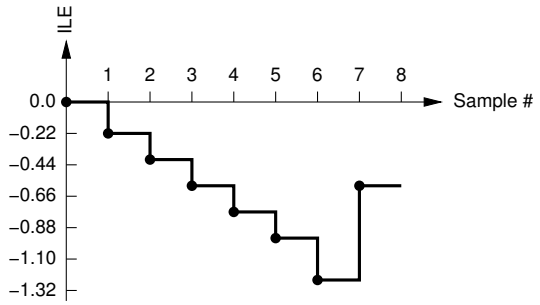
Average count = 9

(a) Graph of DLE function:



(b) DNL is the worst case DLE value (here it is 1.55 LSB.)

(c) Graph of ILE function:



(d) The INL is the worst case ILE value (here it is −1.265 LSB.)

10.13 DAC INL

We assume $V_{max} = 10V$. For the DAC with Δ_i given in Table 10.4 (page 335 of the book):

$$INL = \frac{\sum(+) - \sum(-)}{2B_n}$$

$$\begin{aligned} B_n \text{ (LSB contribution)} &= \frac{V_{max}}{2^n - 1} = \frac{10V}{2^5 - 1} \\ &= 0.3225806452V \end{aligned}$$

$$\Delta_5 = \frac{10V}{32} = 0.3125V$$

and $\Delta_4 = \Delta_3 = \Delta_2 = 0.3125V$ as well, since in an ideal converter, each of these is one quantum voltage. In a real converter, each of these will deviate in different ways from the quantum voltage.

	$(+)$	$(-)$
e_1	$15\Delta_1$	$-\Delta_2 - 2\Delta_3 - 4\Delta_4 - 8\Delta_5$
e_2	$15\Delta_2$	$-8\Delta_1 - \Delta_3 - 2\Delta_4 - 4\Delta_5$
e_3	$15\Delta_3$	$-4\Delta_1 - 8\Delta_2 - \Delta_4 - 2\Delta_5$
e_4	$15\Delta_4$	$-2\Delta_1 - 4\Delta_2 - 8\Delta_3 - \Delta_5$
e_5	$15\Delta_5$	$-\Delta_1 - 2\Delta_2 - 4\Delta_3 - 8\Delta_4$
	$15(\Delta_1 + \Delta_2 + \Delta_3 + \Delta_4 + \Delta_5)$	$-15\Delta_1 - 15\Delta_2 - 15\Delta_3 - 15\Delta_4 - 15\Delta_5$

In the ideal converter, $\Delta_1 = \Delta_2 = \Delta_3 = \Delta_3 = \Delta_4 = \Delta_5$, so $\sum(+) - \sum(-) = 0$, and $\boxed{\text{INL} = 0}$. In a real converter, Δ_i would vary, so INL would not be 0.

10.14 Multi-tone testing

(a)

$$\frac{F_t}{F_s} = \frac{M}{N},\ \Delta = 20Hz,\ M = \frac{F_t}{\Delta} = 101$$

$N = 600$, $F_t = 2,020Hz$

$$F_s = \frac{NF_t}{M} = \frac{600 \times 2,020Hz}{101} = 12,000Hz$$

(b) Choose $\Delta = 10Hz$ to halve the sampling frequency and double the unit test period to 25 *msec*.

Then, with 600 samples, $F_s = N\Delta = 600 \times 10Hz = 6,000Hz$.

Choose $M = 101$, so $F_t = M\Delta = 101 \times 10Hz = 1,010Hz$ and

$$\frac{F_t}{F_s} = \frac{1,010}{6,000} = \frac{M}{N} = \frac{101}{600}$$

(c) FFT frequency bins for second-order harmonics are: $10f,\ 18f,\ 34f$

(d) First-order intermodulation distortion bins: $5f,\ 9f,\ 17f$

Second-order intermodulation bins:
$|5-9| = 4f,\ 5+9 = 14f$
$|5-17| = 12f,\ 5+17 = 22f$
$|9-17| = 8f,\ 9+17 = 26f$

(e) Total harmonic distortion (THD) is the ratio of the energy in the harmonics of the waveform to the energy in the fundamental. Thus,

$$THD = \sqrt{\frac{10^{\frac{H_2}{10}} + 10^{\frac{H_3}{10}} + \cdots + 10^{\frac{H_{10}}{10}}}{10^{\frac{H_1}{10}}}}$$

where H_1 is the amplitude of the fundamental (in dB) and $H_2 \cdots H_{10}$ are the amplitudes of the second through tenth harmonics in dB.

The FFT of the circuit response (the analog output) is taken and the magnitudes in the bins of the harmonics are measured, along with the magnitude in the bin of the fundamental.

Chapter 11: Model-Based Analog and Mixed-Signal Test

11.1 Parametric faults

For the circuit of Figure 11.10,

$$\begin{aligned} V_{out} &= V_{in} + \frac{1}{C + \frac{1}{R_f}} \int_0^t \frac{V_{in}}{R_1} dt \\ &= \frac{V_{in} t}{R_1 C + \frac{R_1}{R_f}} \end{aligned}$$

This is an integrator or low-pass filter, with parameters

f_c - filter cutoff frequency: $C,\ R_f$

A - AC voltage gain: $R_1,\ C,\ R_f$

$A4$ - DC voltage gain: $R_1,\ R_f$

There are no useful single parametric faults, two useful double parametric faults, and one useful triple parametric fault.

11.2 Parametric faults

The circuit of Figure 11.14 contains three stages:

First stage, Buffer: fault $R_1,\ R_2,\ R_8$ (gain)

Second stage, Integrator: fault C_1 (cutoff frequency)
fault $R_2,\ C_1$ (AC voltage gain)
fault R_2 (DC voltage gain)

Third stage, Low-pass filter: fault $C_2,\ R_5$ (cutoff frequency)
fault $R_3,\ C_2,\ R_5$ (AC voltage gain)
fault $R_3,\ R_5$ (DC voltage gain)

DC feedback between first and third stages: fault $R_1,\ R_4,\ R_6,\ R_5,\ R_3$

AC feedback between first and third stages: fault $R_1,\ R_4,\ R_6,\ R_5,\ R_3,\ C_2$

DC feedback between first and second stages: fault $R_1,\ R_4,\ R_8,\ R_2,\ R_7$

AC feedback between first and second stages: fault $R_1,\ R_4,\ R_2,\ R_8,\ C_1,\ R_7$

11.3 Parametric faults

Faults for the circuit of Figure 11.15 are,

1st stage: $R_2,\ C_1$
$R_1,\ R_2,\ C_1,\ R_3$
$R_1,\ R_2,\ R_3$

2nd stage: $R_4,\ R_5,\ R_{13}$

3rd stage: C_2
R_6
$R_6,\ C_2$

4th stage: C_3
$R_7,\ R_{12}$
$R_7,\ C_3,\ R_{12}$

5th stage: $R_8,\ R_9$

6th stage: $C_4,\ R_{10}$
$R_{10},\ R_{11}$
$C_4,\ R_{10},\ R_{11}$

DC feedback between stages 1 and 3: $R_6,\ R_3,\ R_2,\ R_1$

AC feedback between stages 1 and 3: $R_6,\ R_3,\ R_2,\ R_1,\ C_1,\ C_2$

DC feedback between stages 4 and 6: $R_7,\ R_{12},\ R_{10},\ R_{11}$

AC feedback between stages 4 and 6: $C_3, C_4,\ R_7,\ R_{12},\ R_{10},\ R_{11}$

Note: For Problems 11.2 and 11.3, we can simplify the testing of the feedback by computing the composite transfer function. We usually only need to test for the location of the dominant and secondary poles, and for any zeroes that cancel poles (which should also be tested.) These considerations will simplify the testing of multiple parametric feedback faults.

11.4 Parametric faults

For the circuit of Figure 11.16,

1st stage: $R_1,\ R_2$ (DC gain)
$C,\ R_2$ (Roll off f)
$R_1,\ R_2,\ C$ (AC gain)

2nd stage: $R_3,\ R_4$ (DC gain)

3rd stage: $R_6,\ R_5,\ R_7$ (Current summing)

4th stage: $R_8,\ R_9$ (DC gain)

5th stage: $R_{10},\ R_{11}$ (DC gain)
$R_{11},\ C_2$ (Roll off f)
$R_{10},\ R_{11},\ C_2$ (AC gain)

11.5 Transistor parametric faults

For the circuit of Figure 11.17, voltage gain A_v:

$$\boxed{A_v = -g_{M124}(r_{0M125}||r_{0M126}) \times g_{M136}(r_{0M136}||r_{0M135})}$$

So this is a stong function of the Early voltage (V_A) and the bias overdrive conditions.

$$g_m r_0 = \frac{2V_A}{V_{0v}}$$

Faults to test:

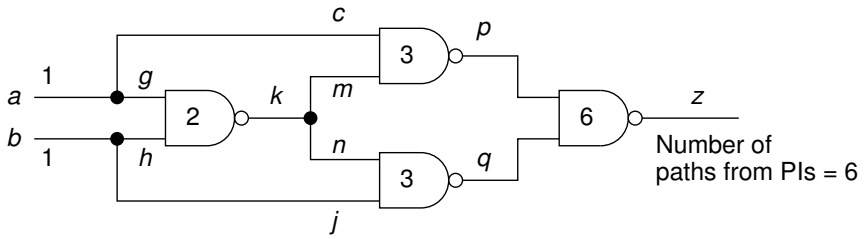
1. $\boxed{K_{M188}/K_{M116}}$ controls current source.
2. $\boxed{K_{M188}/K_{M135}}$ controls current source.
3. $\boxed{R_{144}}$ controls bias current.
4. $\boxed{K_{M124}/K_{M125}}$ controls input offset.
5. $\boxed{K_{M127}/K_{M126}}$ controls input offset.
6. $\frac{\frac{K_{M124}}{K_{M125}K_{M126}}}{\frac{1}{K_{M125}}+\frac{1}{K_{M126}}} = \boxed{\frac{K_{M124}}{K_{M126}+K_{M125}}}$ controls first stage gain.
7. $\frac{\frac{K_{M136}}{K_{M136}K_{M135}}}{\frac{1}{K_{M136}}+\frac{1}{K_{M135}}} = \boxed{\frac{K_{M136}}{K_{M135}+K_{M136}}}$ controls second stage gain.
8. $\boxed{\frac{1}{K_{M136}C_1}}$ dominant pole location.
9. $\boxed{\frac{1}{\left(\frac{1}{K_{M136}}-R_2\right)C_1}}$ zero location.

Chapter 12: Delay Test

12.1 Non-robust path-delay test

The given circuit has no redundant single stuck-at fault. This can be verified either by an ATPG program or by manually simulating all four input vectors.

The circuit has six paths. The following figure illustrates path counting. Each PI or gate is assigned a label that gives the number of paths from all PIs. Labels of PIs are 1. The label of a gate is the sum of labels of its fanins. The label of the output gate gives the total number of paths.



Labels show number of paths from primary inputs.

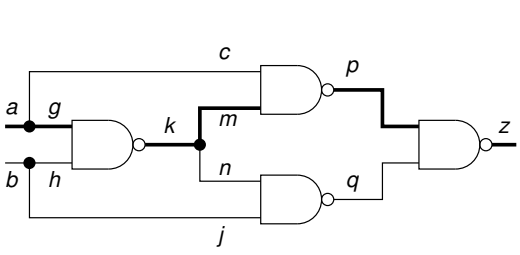
Eight tests and the singly-testable (nonrobustly testable) path-delay faults (PDFs) detected by them are listed in the following table. We note that the non-robust detection of a PDF requires an input transition and a statically sensitized path by the second vector of the two-vector test.

Test	Detected PDFs
$a = R1,\ b = S0$	$\uparrow a - c - p - z$
$a = R1,\ b = S1$	$\uparrow a - g - k - n - q - z$ and
	$\uparrow a - g - k - m - p - z$
$a = F0,\ b = S0$	$\downarrow a - c - p - z$
$a = F0,\ b = S1$	$\downarrow a - g - k - n - q - z$
$a = S0,\ b = R1$	$\uparrow b - j - q - z$
$a = S1,\ b = R1$	$\uparrow b - h - k - m - p - z$ and
	$\uparrow b - h - k - n - q - z$
$a = S0,\ b = F0$	$\downarrow b - j - q - z$
$a = S1,\ b = F0$	$\downarrow b - h - k - m - p - z$

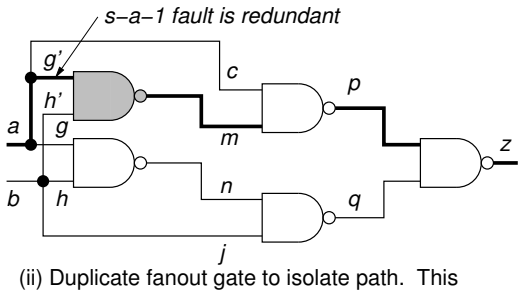
Two singly-untestable PDFs are:

1. $\downarrow a - g - k - m - p - z$
2. $\downarrow b - h - k - n - q - z$.

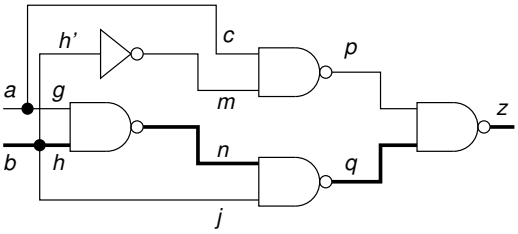
Elimination of untestable PDFs: (This part may be expected only from a student of an advanced course.) The procedure in the next figure illustrates the KMS algorithm, which results in a fully testable circuit. See reference [352] of the book.



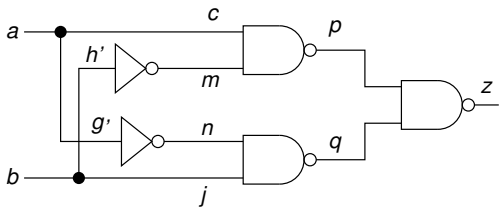
(i) Identify untestable PDF a–g–k–m–p–z. All single stuck–at faults are testable.



(ii) Duplicate fanout gate to isolate path. This circuit is functionally unchanged and has the same number (6) paths. Fault s–a–1 on g' is redundant.



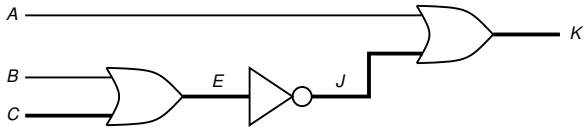
(iii) Eliminate g' s–a–1. Function is unchanged, PDF a–g–k–m–p–z is eliminated, and of the remaining 9 PDFs, b–h–n–q–z is untestable.



(iv) Apply the same procedure to eliminate the untestable PDF. All 8 PDFs in this XOR circuit are testable.

12.2 Robust path-delay tests

To remove the redundant fault Q s-a-1 in the circuit of Figure 12.2, line Q is set to 1, all implied signals are also set, and any gates and signal having no effect on the primary output are removed. For details of this procedure one may refer to Chapter 7. The resulting circuit is shown below.



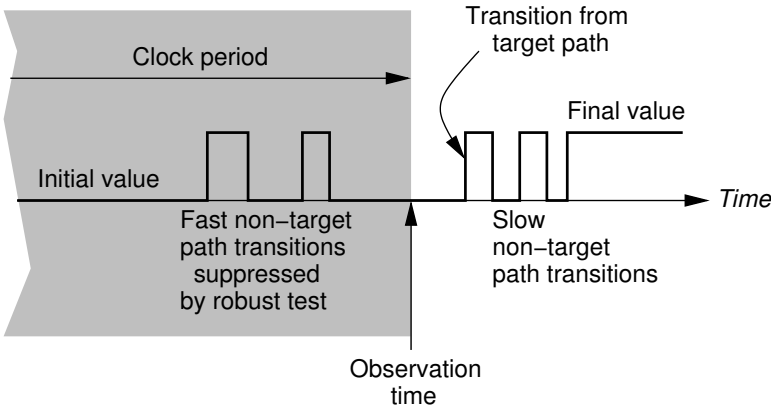
The circuit now has three paths. For each path, all off-path inputs can be directly controlled from PIs. For example, consider the path, $C - E - J - K$, shown with bold lines. We can set off-path inputs as $B = S0$ and $A = S0$. Now, applying a rising or a falling transition at C will robustly test the path for the corresponding transition. A similar argument applies to the other two paths.

Note: This is a fanout-free circuit. It has exactly one path between each PI-PO pair. Each path has two single input change (SIC) test vector pairs that are robust tests for the path.

12.3 Robust path-delay tests

According to Definition 12.3 (see page 422 of the book), a robust path-delay test must produce an observable output value that is different from the correct output

whenever the delay of the path exceeds the observation instant (usually the clock boundary. A general output waveform is shown in the figure.



A robust test example showing a failing path.

Each path to the output can potentially produce a transition, whose time of occurrence depends on the delay of the path. By properly setting the off-path values a robust test suppresses all "fast transitions." Thus, the transition arriving through the target path is the first transition to appear at the output. If the target path is faulty then the output value observed will be the "initial value" (0 in the figure.) To be discriminated with the correct (or expected) output value, this must be different from the initial value. ■

Notice that the other slow transitions can make the test to show a failure even when the target path is not faulty. But they can never make the test to pass when the target path is faulty. In general, a robust test only guarantees detection and not diagnosis.

The circuit of Figure 12.4 cannot have a real transition at the output since the steady-state logic value is always 0. Thus, no robust test is possible for any path in this circuit.

12.4 Single-input change (SIC) tests

Consider a two-vector input sequence $(V1, V2)$ applied to a combinational logic circuit. For non-robustly testing an input to output path, the sequence should satisfy two conditions:

1. *Static sensitization* – the second vector $V2$ must sensitize the entire path.

2. *Transition at the origin* – the two vectors must produce a signal transition at the origin of the path.

If $(V1, V2)$ is a single-input change (SIC) vector-pair, such that

- $V2$ sensitizes a target path with path origin at 1(0) for a rising (falling) transition to be propagated, and

- $V1$ is same as $V2$ except that the bit at the path origin is flipped,

then the $(V1, V2)$ vector sequence satisfies the two conditions for a non-robust test listed above. ■

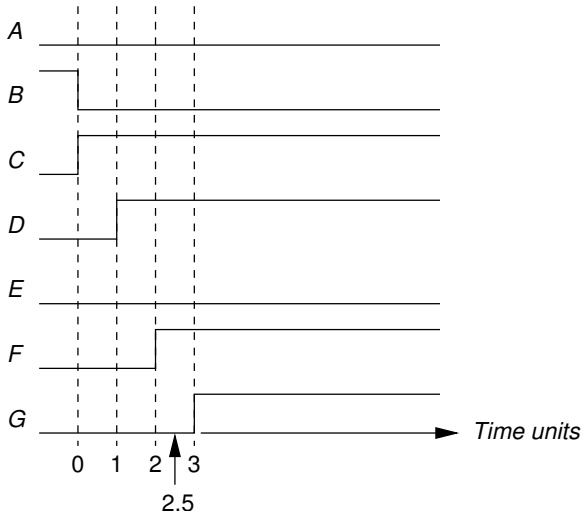
Note: If the circuit is free from fanouts, then the input change in $V1$ can only affect the signals on the path. As a result, all off-path signals will remain steady ($S0$ or $S1$) during both vectors and the path will remain sensitized. This is an over specification of the conditions required for a robust test. Thus, for a fanout free circuit, there exists a robust path-delay test for every path that is statically sensitizable. Also see Problem 12.2.

12.5 Path-delay tests

(a) The required test for path $\uparrow C - F - G$ in Figure 12.14(a) (page 437 of the book) is $A = S0,\ B = U0,\ C = R1$.

(b) Yes, the test will work because a falling transition at B does not violate the $B = U0$ requirement of the test in (a).

(c) The waveforms for the circuit of Figure 12.14(a) for the test in part (b) above are sketched below. The output rises after three units of time and will have an incorrect value of 0 at 2.5 units. This test propagates transitions through two paths, $\downarrow B - D - F - G$ and $\uparrow C - F - G$. Any one or both can be faulty. A diagnosis is not possible with this test.

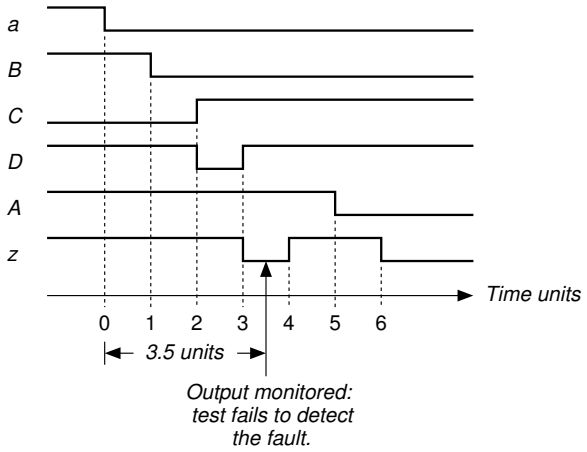


(d) To diagnose the faulty path, we apply four tests:

1. $A = S0,\ B = S0,\ C = R1$ *and* $F0$ will test the paths $\uparrow C - F - G$ and $\downarrow C - F - G$, respectively.
2. $A = S0,\ B = R1$ *and* $F0,\ C = S1$ will test the paths $\uparrow B - D - F - G$ and $\downarrow B - D - F - G$, respectively.

12.6 Path-delay test robustness

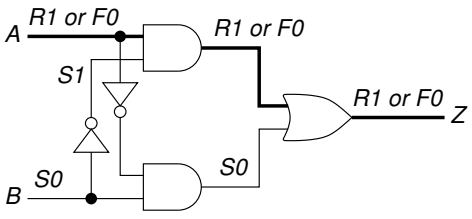
(a) Waveforms for the circuit in Figure 12.14 (b) are sketched below. When the



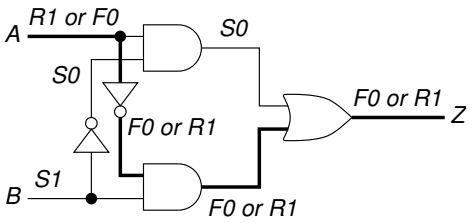
output z is monitored 3.5 time units after the application of the falling transition at a, we observe a correct value (0), although the target path $a - A - z$ has a delay fault. Note that path $a - C - D - z$ is also faulty and interferes with the testing of the target path. The given input is a non-robust test and, by definition, is only guaranteed to work if the target path is the only faulty path.
(b) A robust test will require $D = S1$, which cannot be justified since a must be set to $F0$ to activate the target path. Thus, **a robust test is impossible**.

12.7 Off-path signals

Consider an exclusive-OR (XOR) circuit, implemented with Boolean gates as shown in the following figure. There are two paths from input A to output Z. The non-inverting path $A \rightarrow Z$ is tested by setting the off-path input B to a steady 0 ($B = S0$) for any transition ($R1$ or $F0$) at A. A test for the inverting path $A \rightarrow Z$ is tested by setting $B = S1$.



Tests for a non–inverting path.



Tests for an inverting path.

Thus, the off-path input of an XOR circuit should be set to a steady value. If it is set to $S0$, then the output transition will be of the same type as the on-path input. If the off-path input is set to $S1$, then the output transition will be an inversion of the on-path input. In general, one might assume that the inverting path would have greater delay (three gates vs. two gates.)

12.8 Logical and timing conditions

(a) Logical Condition: For Z to follow A, we must set B, C, *etc.*, to non-controlling values of the Boolean gate when A has the non-controlling value, or B, C, *etc.*, can have any arbitrary values (don't care) when A has the controlling value.

(b) Timing Condition: When the pre-transition state of A is the non-controlling value for the Boolean gate then B, C, *etc.* must be set to non-controlling values, which should remain steady across the transition on A. When the pretransition state of A is the controlling value for the Boolean gate then B, C, *etc.*, can have any arbitrary values (don't care).

Using these two conditions, off-path signal values can be obtained for propagating the delay test signal through A. For example, consider an AND gate. A rising transition at A will require all off-path signals to be $U1$. A falling transition at A will require all off-path signals to be $S1$.

12.9 Path counting

We consider the combinational circuit as a directed graph with PIs, POs and gates as vertices, and the arcs drawn according to connectivity. We add two vertices, a vertex named *source* from which arcs are directed to all PI vertices, and a *sink* vertex to which arcs are directed from all PO vertices. Each vertex v is given a label, $N(v)$, whose value denotes the number of paths from *source* to v. The path counting algorithm is as follows:

1. *Initialization:* Set all labels to 0. Update $N(source) = 1$.

2. *Count:* Update each vertex only after all of its fanin vertices have been updated. Update of vertex v is done as follows:

$$N(v) = \sum_{i=1}^{k} N(v_i)$$

 where $v_1, v_2, \ldots, v_k$ are the fanin vertices of v.

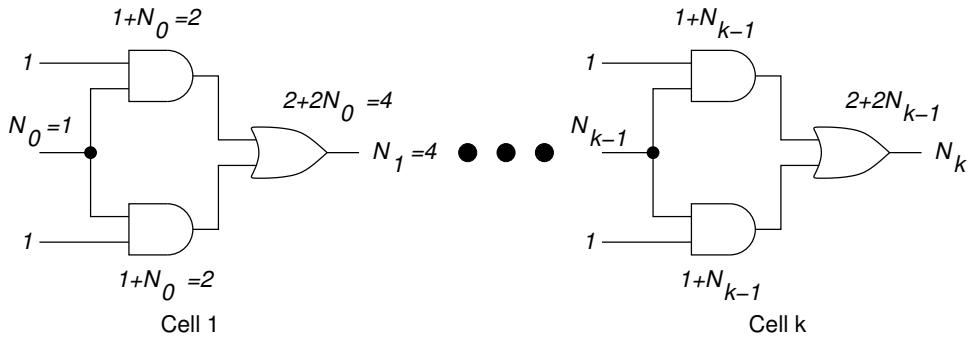
3. *Result:* $N(sink)$ = number of paths in the circuit.

Complexity: Since each vertex is processed once, there are n updates, where n is the number of vertices in the graph. Each update requires adding the labels of the fanin vertices. An upper bound on fanin is n. Thus, the complexity of the path counting algorithm is $O(n^2)$, where $n = PI + PO + gates + 2$. In general, however, the fanin of a gate does not grow with the number of gates, and the complexity remains closer to $O(n)$.

12.10 Pomeranz-Reddy example

In the following figure nodes (PIs and gate outputs) are labeled by the number of paths between the node and all PIs. Thus, all PI labels are 1. The output label of *ith* cell is shown as N_i. Path counting proceeds from left to right. The label of a node is determined as the sum of the fanin node labels. Thus, $N_0 = 1$, $N_1 = 4$, and

$$N_k \;=\; 2(1 + N_{k-1})$$



Using recursion, we obtain

$$\begin{aligned}
N_k &= 2(1 + 2(1 + N_{k-2})) \\
&= 2(1 + 2 + 2N_{k-2}) \\
&= 2(1 + 2 + 2^2(1 + N_{k-3})) \\
&= \;\; . \; . \; etc. \; . \; . \\
&= 2(1 + 2 + 2^2 + 2^3 + 2^4 + \; \ldots \; + 2^{k-1}(1 + N_0)) \\
&= 2(1 + 2 + 2^2 + 2^3 + 2^4 + \; \ldots \; + 2^{k-1}) + 2^k, \text{ since } N_0 = 1 \\
&= 2(2^k - 1) + 2^k = 3 \times 2^k - 2
\end{aligned}$$

which is the desired result. ■

12.11 Sequential path-delay fault testing

A robust test for the path *d-e-f-g* consists of a vector-pair that must satisfy two necessary conditions:

1. The values of inputs a, b and c are set in such a way that any change at the path destination g must be preceded by a change at the path origin d.

2. A transition is applied at the path origin d.

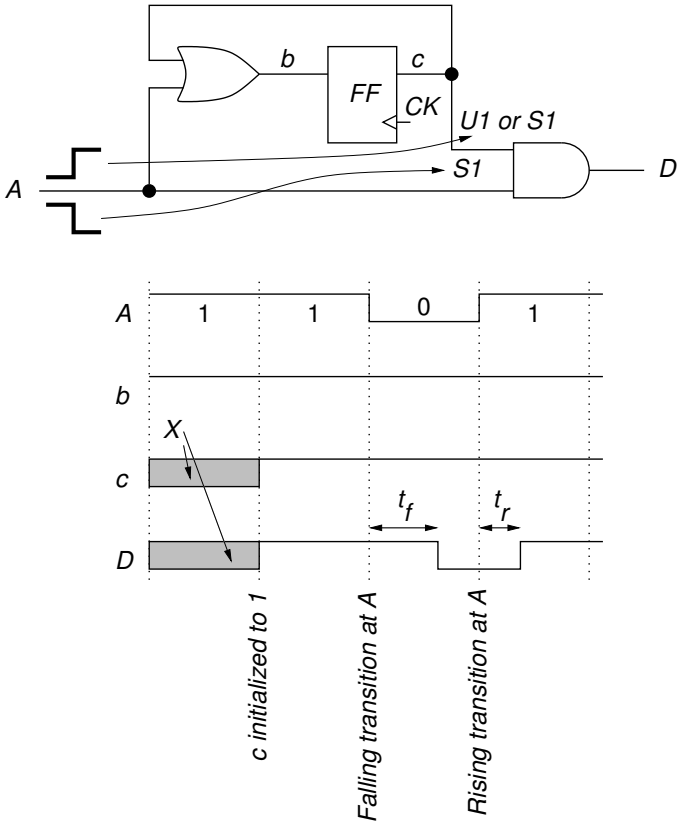
In this case, however, g is the next state for the input d. So any change in d must be preceded by a change in g. Therefore, as long as a, b and c satisfy the robust test condition, a transition in d awaits a transition in g, which awaits a transition in d.

This cyclic situation makes it impossible to create a transition at the path origin. Hence, no robust test is possible for this path. ■

Note: This result can be generalized – a path is rubustly untestable if it has an even number of inversions and its destination feeds back into the origin through a single clocked flip-flop. Such a path need not be a false path and can often be tested by a non-robust test.

12.12 Sequential path-delay fault ATPG

The states of signal c for the two tests are: $c = U1$ *or* $S1$ for $\uparrow A - D$ and $c = S1$ for $\downarrow A - D$. We first initialize c to 1 by applying $A = 1$ and clocking the flip-flop. Now the state of c will remain 1 irrespective of the signal value at A. Thus, $A = 1101$ will robustly test both faults. The following figure illustrates the test, where t_f and t_r are the fall and rise delays, respectively, of the path $A - D$.



Chapter 13: IDDQ Test

13.1 Leakage fault tests

Examining Figure 13.12(c), we see that rows $i = 1$, 2, and 4 of the leakage fault table cover all possible tests. We need these stuck-fault vectors:

I_1	I_2	O_1	
0	0	1	$i = 1$
0	1	0	$i = 2$
1	0	0	$i = 4$

13.2 Tester time

Total tester cost is $512 \times \$9k = \$4,608,000$. We need to make $0.02 \times 100,000 = 2,000$ I_{DDQ} measurements. At $2\mu A$ resolution, 335.4 msec are needed to make an I_{DDQ} measurement.

$$\begin{aligned} \text{Test time} &= \frac{100,000 - 2,000}{750MHz} + 2,000 \times 335.4\ msec/vector \\ &= 670.8\ sec \end{aligned}$$

$$\begin{aligned} \text{Total ATE cost over 10 years} &= \$4,608,000 + 10 \times \$50,000 \\ &= \$5,108,000 \end{aligned}$$

$$\begin{aligned} \text{ATE cost/sec} &= \frac{\$5,108,000}{10 \times 365 \times 24 \times 60 \times 60} \\ &= \$0.0162/sec \end{aligned}$$

$$\begin{aligned} \text{Test cost/chip} &= \$0.0162/sec \times 670.8\ sec \\ &= \$10.87/chip \end{aligned}$$

13.3 I_{DDQ} threshold

To reduce the functional field failures to 1%, the I_{DDQ} limit should be below $50\mu A$ according to Figure 13.16. We set the I_{DDQ} threshold to $40\mu A$.

$$\begin{aligned} \text{Total cost} &= \text{Total cost} + \text{Field return cost} \\ &= \#\ \mu\text{processors} \times \text{Test cost}/\mu\text{processor} \\ &\quad + \text{Failure rate} \times \text{Field return cost} \end{aligned}$$

Thus, without an I_{DDQ} test,

$$\begin{aligned} \text{Test cost} &= 20 \times 10^6 \times \$40 + 0.01 \times 20 \times 10^6 \times \$300 \\ &= \$800\ million + \$60\ million\ =\ \$860\ million \end{aligned}$$

With stringent I_{DDQ} test such that there are no field failures,

$$\text{Test cost} \quad = \quad 20 \times 10^6 \times \$55 = \$1.1 \; billion$$

We are better off shipping defective product. Improved I_{DDQ} test adds \$300 million to the testing costs while saves only \$60 million on the field return costs.

13.4 Built-in current testing

$$\#\text{ n wells} = \frac{\frac{50\times10^6 \; transistors}{2}}{50} = 500,000$$

$$\begin{aligned}
\text{Total leakage current} &= \#\text{ n wells} \times \text{n well leakage} \\
&\quad +\#\text{ transistors (drain leakage} + \text{subthreshold leakage)} \\
&= 500,000 \times 0.5\mu A + 50,000,000(0.01nA + 0.005nA) \\
&= 0.625mA
\end{aligned}$$

Assume that maximum current/BIC can be $2.5\mu A$. Then,

$$\begin{aligned}
2.5\mu A &= \frac{\#\text{ transistors}}{100} \times 0.5nA + \#\text{ transistors} \times 0.015nA \\
&= \#\text{ transistors} \times 2 \times 10^{-11}A \\
\#\text{ transistors} &= \frac{2.5\mu A}{2 \times 10^{-11}A} = 125,000
\end{aligned}$$

At most 62,500 nFET-pFET pairs can be handled by one BIC sensor. The number of BIC sensors for a partitioned ground bus is given by,

$$\frac{0.625mA}{2.5\mu A} = 250 \; sensors$$

Assume that the n-well substrate boundary leaks all the time. Assume that transistors are on half the time (50% duty cycle.)

13.5 Built-in current testing

$$\boxed{\#\text{n wells} = \tfrac{50\times10^6}{2\times30} = 833,334}$$

$$\begin{aligned}
\text{Total leakage current} &= \#\text{n wells} \times 0.55nA + \#\text{transistors}(0.011nA + 0.006nA) \\
&= 833334 \times 0.55nA + 50 \times 10^6 \times 0.017nA \\
&= \boxed{1.309\text{mA}}
\end{aligned}$$

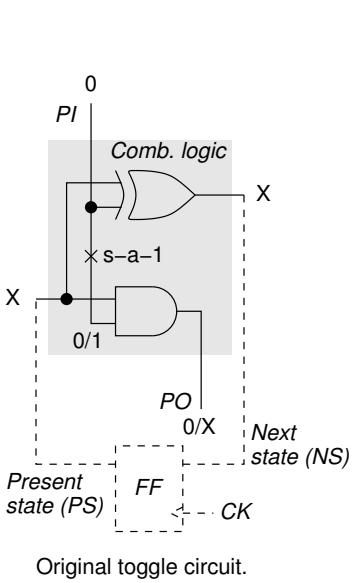
Load up 1 BIC sensor with only $1.5\mu A$ of leakage current (for a safety margin.)

$$\text{Total \# of sensors} = \frac{1.309mA}{1.5\mu A} = \boxed{873 \text{ sensors}}$$

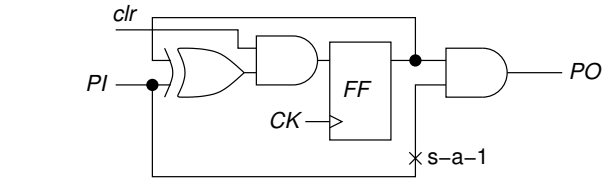
Chapter 14: Digital DFT and Scan Design

14.1 Importance of initialization

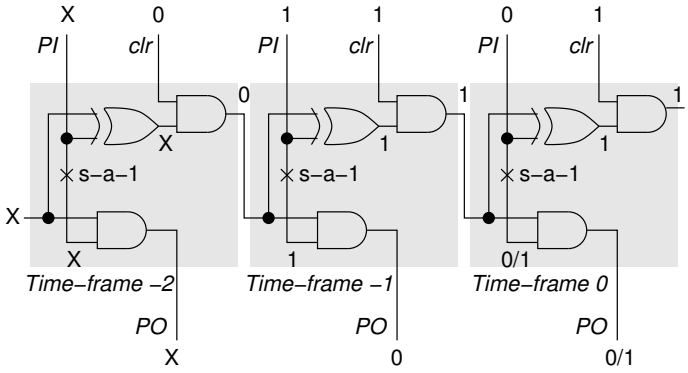
The left diagram in the following figure shows the test generation for the given s-a-1 fault in the toggle circuit. The present state (PS) is X and so is the next state (NS), irrespective of the signal value at the primary input (PI). The fault is activated by setting $PI = 0$. This produces an output $PO = 0/X$. Thus, the fault is potentially detected.



Original toggle circuit.



Toggle circuit with initialization input, *clr*.



Test generation for toggle circuit with initialization input.

We notice that the fault can be activated as 0/1, but to observe its effect we must have the present state (PS) as 1. Since no input can initialize the circuit, PS always remains X and the fault can only be potentially detected. Use of an ATPG program will show that only the PO s-a-1 can be deterministically detected. Two faults, PI s-a-1 and the one shown in the figure, are potentially detected by $PI = 0$ input. All other faults are untestable.

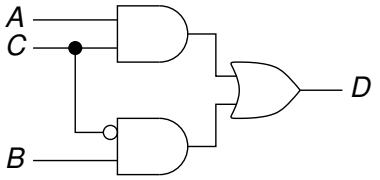
A possible design change is shown in the top right diagram. We add an initialization input clr. When $clr = 0$, the FF output is forced to 0. For $clr = 1$ the circuit functions as the original toggle circuit. As shown in the lower right diagram, a test for the given s-a-1 fault is obtained in three time-frames. The first vector, $PI = X,\ clr = 0$, initializes the circuit as $NS = 0$. The second vector, $PI = 1,\ clr = 1$, toggles the state to $NS = 1$ in time-frame -1. This is the required state for testing the fault. Third vector, $PI = 0,\ clr = 1$, activates the fault, whose effect is propagated to PO as 0/1.

Use of an ATPG program on the toggle circuit with the initialization input will show that all faults are deterministically detectable with the exception of one fault. That fault, clr s-a-1, is potentially detectable.

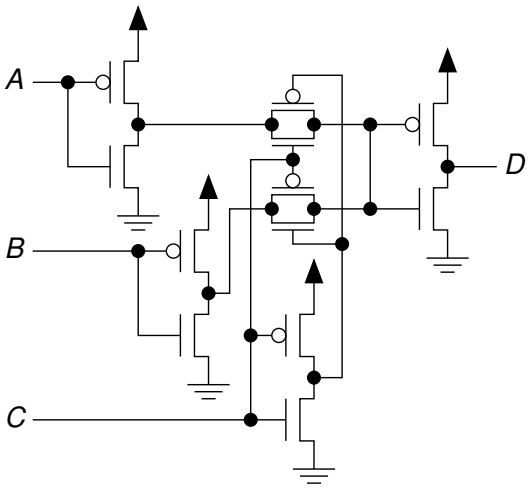
14.2 MUX design

Note: The statement of this problem should be revised to read, "Design an economical CMOS circuit for a static two-to-one multiplexer."

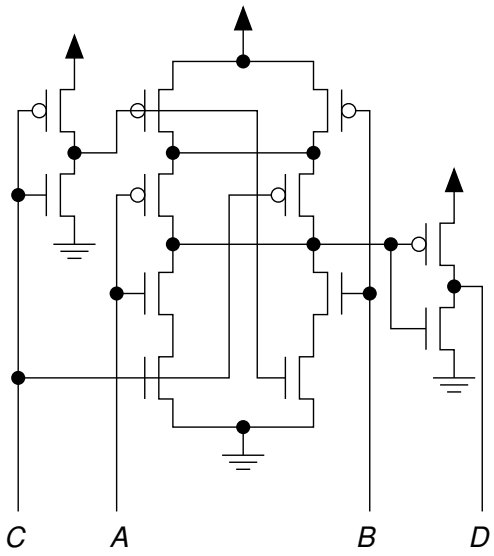
The following figure shows two static designs of the multiplexer function shown in (a). A requirement of a static design is that incoming signals should not be connected to transistor channels. Design (b) uses CMOS transmission gates. Static inverters in signals A and B provide isolation between the two input signals. The output inverter cancels the inversion. Note that if inverters in A and B, and the



(a) Logic function of multiplexer.



(b) A static design with transmission gates.



(c) A static design with only complementary gates.

output inverter, are eliminated, the circuit will still provide the multiplexer function. In that multiplexer, which will require only six transistors, a path between the inputs A and B can be created momentarily if there is a time delay between the signals C and $\overline{C}$. Such a path can sometimes upset the states of the flip-flops that supply A and B signals.

The design (c) uses only complementary CMOS gates. Both designs (b) and (c) require 12 transitors.

14.3 Multiple scan chains

Note: *Please observe the corrections to the first printing of the book as posted at*

the website. These corrections are included in second printing of 2001.[1]

Number of flip-flops in each chain = $\frac{n_{sff}}{n_{chain}}$ and, therefore, Equation 14.1 will change to:

$$\text{Scan test length} = \frac{n_{sff}}{n_{chain}} + 4 + (\frac{n_{sff}}{n_{chain}} + 1) \times n_{comb} + \frac{n_{sff}}{n_{chain}}$$

To compute the gate overhead, we notice that we will need to add a MUX at each scan chain output to multiplex the scan and the normal output. Input pins will not require any additional MUXes as the MUXes added at the first flip-flop of each scan chain can be used to multiplex the corresponding input also. Hence,

$$\text{Gate overhead} = \frac{4 \times n_{chain} + 4 \times n_{sff}}{n_g + 10 \times n_{ff}} \times 100\%$$

Solution provided by K. K. Saluja

14.4 Scan tests

Assume 20 equal length scan chains, each having 2000/20 = 100 flip-flops. Scan sequence test length is given by:

$$\begin{aligned} \text{Scan test length} &= (n_{comb} + 2) \times n_{chain} + n_{comb} + 4 \\ &= (500 + 2) \times 100 + 500 + 4 \\ &= 50,704 \text{ clock cycles} \end{aligned}$$

where n_{comb} = number of combinational vectors, and n_{chain} = number of flip-flops in the longest scan chain.

Gate Overhead: All scanin inputs are obtained as fanouts of normal PIs. A multiplexer is inserted between each PO and its normal output signal. The other data input of the multiplexer is a scanout and control is the test control (TC) PI. Assuming normal data flip-flops of 10 gates, the overheads are:

$$\text{Overhead (single chain)} = 4n_{sff} + 4 = 4 \times 2000 + 4 = 8,004 \text{ gates}$$

where n_{sff} = total number of scan flip-flops. A multiplexer is assumed to have 4 gates. Only one multiplexer is added for the scanout.

$$\text{Extra overhead (20 chains)} = 4(N_{chain} - 1) = 4 \times 19 = 76$$

[1] Equation 14.1 should read as:

$$\begin{aligned} \text{Scan test length} &= n_{sff} + 4 + (n_{sff} + 1)n_{comb} + n_{sff} \\ &= (n_{comb} + 2)n_{sff} + n_{comb} + 4 \end{aligned}$$

and Equation 14.2 should read as:

$$\text{Gate overhead of scan} = \frac{4 \times n_{sff}}{n_g + 10 \times n_{ff}} \times 100\%$$

$$\text{Total gates in pre} - \text{scan circuit} \quad = \quad 100,000 + 10n_{ff} = 120,000$$

where n_{ff} = total number of flip-flops.

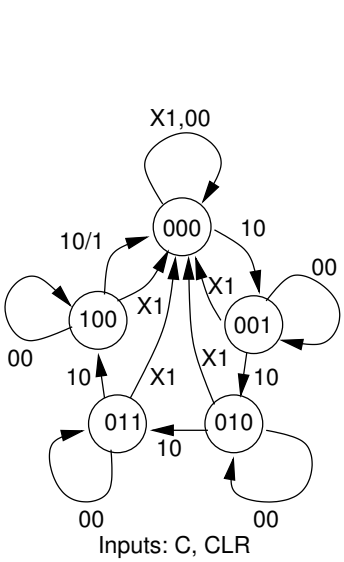
$$\text{Extra overhead of 20 chains} = \frac{76}{120,000 + 8,004} \times 100 = 0.06\%$$

An overhead of 0.06% is incurred to reduce the test length by a factor of $\sim$ 20.

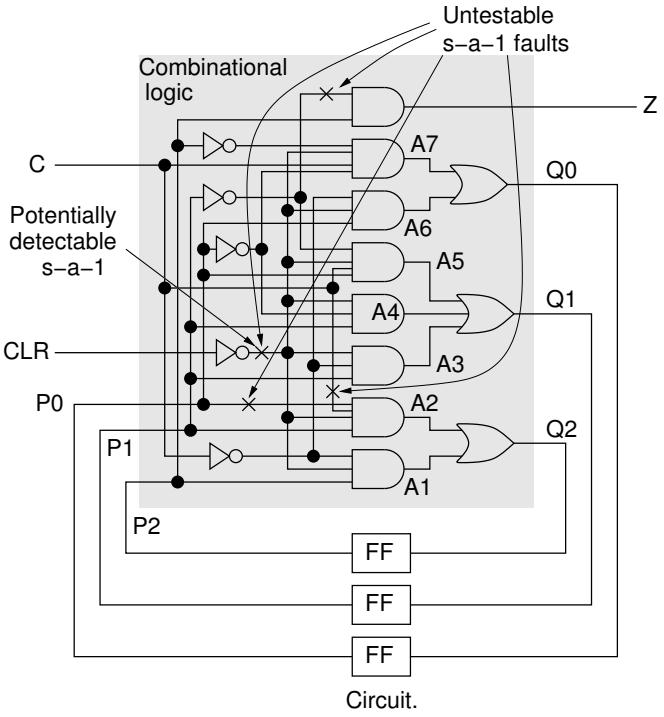
Solution provided by K. K. Saluja

14.5 Modulo-5 counter circuit

The following figure shows a modulo-5 counter circuit. As shown in the state diagram, the states are encoded as 000, 001, 010, 011 and 100. The input $CLR = 1$ initializes the circuit to the 000 state. Input $C = 1$, $CLR = 0$ advances the state at every clock. The clock signal applied to the three D flip-flops is not shown.



State diagram.



Modulo–5 counter.

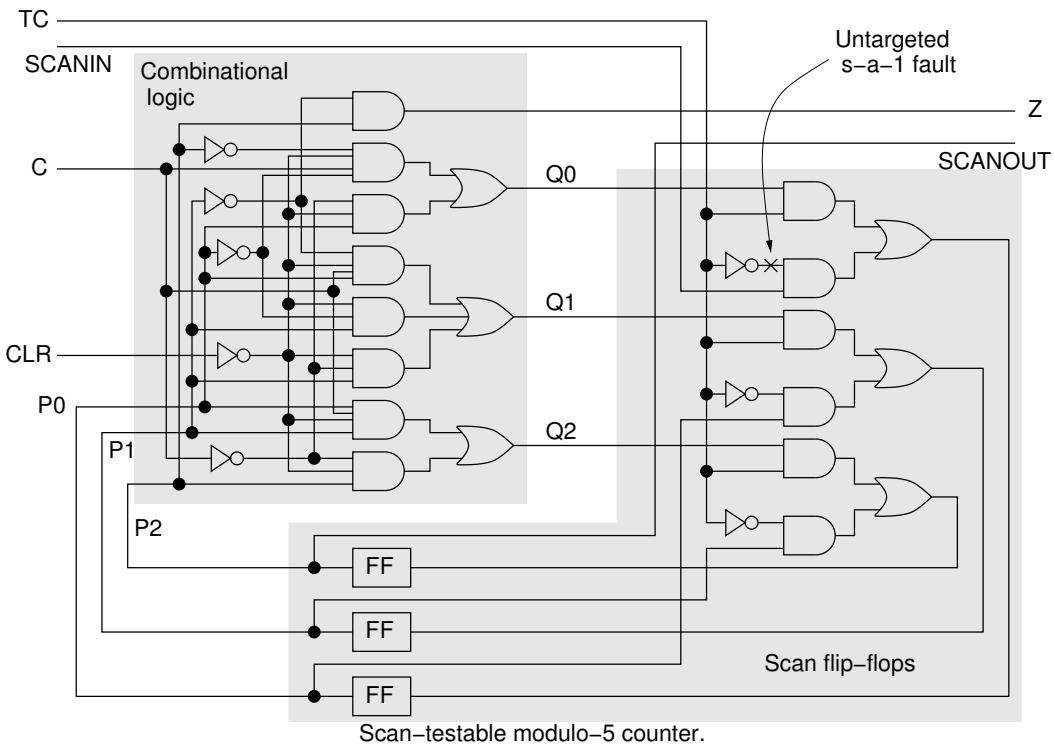
The output Z remains 0 with the exception of the state 100, which produces a $Z = 1$ output.

The combinational circuit (shown in the grey box) is made completely single-fault testable by removing redundant faults that were identified by an ATPG program.

For the sequential counter, a sequential circuit ATPG program produced 62 vectors to obtain a coverage of $(57/62) \times 100 = 92.98\%$. The five untestable faults were all s-a-1 type and are shown in the figure. Among these the s-a-1 fault on the CLR signal was potentially detected by the test set.

14.6 Full-scan design

The following figure shows the scan design of the modulo-5 counter.



The number of vectors obtained may vary depending on the ATPG program used. These results were obtained from Bell Labs' Gentest program. The combanitional circuit, whose inputs are C, CLR, $P0$, $P1$ and $P2$, and outputs are Z, $Q0$, $Q1$ and $Q2$, has a collapsed set of 57 faults. All of these faults were detected by 16 vectors.

A complete scan sequence consists of 74 vectors (see Equation 14.1 in the book), which includes 7 vectors for testing the scan register. The scan circuit contains a collapsed set of 79 faults. Fault simulation of the 74-vector sequence showed that 78 faults were detected. The undetected s-a-1 fault is marked on the circuit diagram. It is at the output of the test control (TC) inverter in the first multiplexer.

The reason this fault is not detected is that it was never targeted. Since the scan register test holds TC to 0 for a continuous scan mode, this fault was not activated. The fault is, however, activated every time the circuit is set in the normal mode during the application of the scan sequence. Since in the normal mode the state of $SCANIN$ is considered irrelevant, $SCANIN$ was arbitrarily set to 0. That prevented the propagation of the fault effect. A suitable strategy for detecting this fault is to set $Q0$ outputs of the combinational logic as 0 by applying $CLR = 1$. At the same time, the circuit is set in the normal mode by applying $TC = 1$. The fault effect is now propagated to the flip-flop and can be scanned out.

We notice that similar faults in the other two multiplexers were detected by our scan sequence. This is due to the chance occurrence of normal data as 0 and scan

data as 1 when $TC = 1$, which would place the fault effect in the flip-flop. $TC = 1$ was always followed by scanout that detected the fault.

In general, it can be recommended that $SCANIN$ is set to 1 whenever the circuit goes to the normal mode (TC=1), provided the AND-OR type of multiplexer is used.

14.7 Scan overhead

If k tracks per routing channel are used for scan routing, then we first modify Equation 14.4 as,

$$Y' = Y + kry$$

where Y is the height of the non-scan chip, r is the number of routing channels, and y is the track width (i.e., vertical space occupied by a horizontal wire.) Substituting $r = Y(1-\beta)/(yT)$, where β is the routing fraction of the total chip area and T is the cell height as a multiple of the track width y, we obtain

$$Y' = Y + \frac{kY(1-\beta)}{T}$$

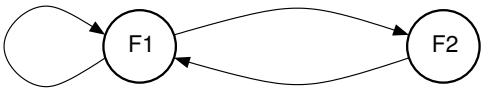
Now the area overhead, which was expressed by Equation 14.5, changes to

$$\begin{aligned} \text{Area overhead of scan} &= \left[(1+\alpha s)\left\{1+\frac{(1-\beta)k}{T}\right\}-1\right]\times 100\% \\ &\approx \left[\alpha s + \frac{(1-\beta)k}{T}\right]\times 100\% \end{aligned}$$

where α is the fractional width increase of a scan flip-flop over a non-scan flip-flop cell, and s is the fraction of the total cell area occupied by the flip-flop cells in the non-scan chip.

14.8 Partial-scan

The s-graph of the circuit in Figure 14.16 is given below.



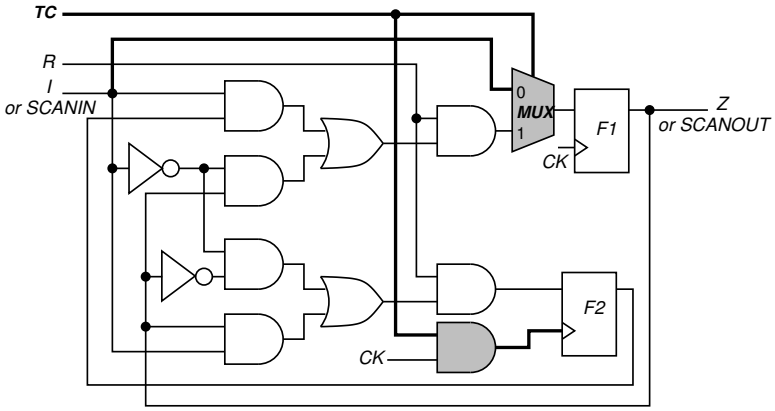
s-graph for the circuit of Figure 14.16.

By scanning F1 all cycles can be eliminated.

14.9 Partial-scan

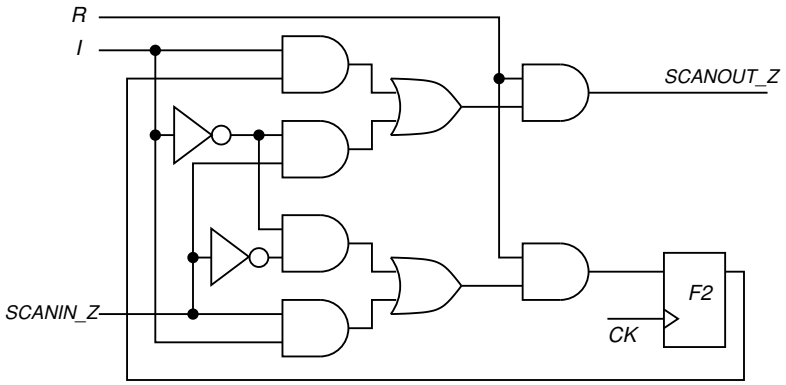
The partial-scan circuit is given below. Added circuitry is shown in grey and wiring, in bold lines. We insert a multiplexer at the input of F1. One input of this multiplexer is the normal input of F1. The other input is a fanout of PI I, which is now also used as $SCANIN$. The control input of the multiplexer is a new PI, TC.

$TC = 0$ is scan mode and $TC = 1$ is the normal mode. $TC = 0$ also inhibits the clocking of the non-scan flip-flop $F2$ such that it holds its state during the scan operation. This is accomplished by using the grey-shaded AND gate. PO Z also acts as $SCANOUT$.



Partial-scan design of the circuit of Figure 14.16.

The ATPG circuit is obtained by removing the MUX and $F1$, making Z a new PI $SCANIN_Z$, and making the AND gate output feeding into the MUX a new PO $SCANOUT_Z$. This circuit is shown in the next figure. A sequential circuit ATPG

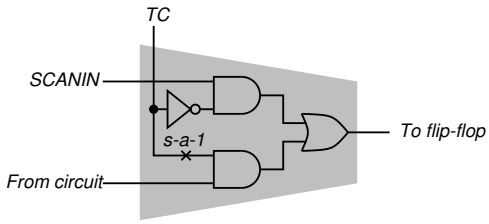


ATPG circuit for the partial-scan design of the circuit of Figure 14.16.

program, GENTEST[2], produced 11 vectors to detect all faults in this circuit. These vectors were converted into scan sequences (see Chapter 14 of the book.) Thus, a set of 28 vectors was produced, which also includes 5 vectors for testing the scan register. The following table shows the test sequence. When the partial-scan circuit was simulated in the sequential mode, these 28 vectors detected all faults, except one fault that was potentially detected. That fault was a s-a-1 fault in the MUX circuit and is shown in the next figure. This happened because we left the input R in the unknown state (X) during the scan mode. If $R = 0$ was used instead, the s-a-1 fault in the MUX would not be detected. However, if $R = 1$ was used, then that

[2]Any other sequential ATPG program can also be used. See Chapter 8 of the book.

fault would have been detected. The detection of such faults is not guaranteed since they are not targeted by the ATPG. Being a part of the scan structure, the MUX is not included in the ATPG circuit. This is a typical situation for scan design.



Potentially detected fault in the scan multiplexer.

Test sequence for partial-scan design of circuit of Figure 14.16.				
Vector number	Input bits TC	R	I	Functions performed
1	0	X	0	Vectors 1 through 5 test scan
2	0	X	0	register in scan mode ($TC = 0$).
3	0	X	1	They apply a 0011 bit stream to
4	0	X	1	I and observe it at Z.
5	0	X	0	
6	0	X	0	Scan-in 0
7	1	0	0	Apply 00 to R and I in normal mode
8	0	X	1	Scan-out Z and scan-in 1
9	1	1	1	Apply 11 to R and I in normal mode
10	0	X	0	Scan-out Z and scan-in 0
11	1	1	0	Apply 10 to R and I in normal mode
12	0	X	1	Scan-out Z and scan-in 1
13	1	1	0	Apply 10 to R and I in normal mode
14	0	X	0	Scan-out Z and scan-in 0
15	1	1	0	Apply 10 to R and I in normal mode
16	0	X	0	Scan-out Z and scan-in 0
17	1	1	1	Apply 11 to R and I in normal mode
18	0	X	1	Scan-out Z and scan-in 1
19	1	1	0	Apply 10 to R and I in normal mode
20	0	X	1	Scan-out Z and scan-in 1
21	1	1	1	Apply 11 to R and I in normal mode
22	0	X	0	Scan-out Z and scan-in 0
23	1	1	1	Apply 11 to R and I in normal mode
24	0	X	0	Scan-out Z and scan-in 0
25	1	1	1	Apply 11 to R and I in normal mode
26	0	X	1	Scan-out Z and scan-in 1
27	1	0	0	Apply 00 to R and I in normal mode
28	0	X	0	Scan-out Z and scan-in 0

14.10 Partial scan

Suppose we arbitrarily select one non-scan flip-flop and scan all other flip-flops. Since there are no self-loops in the original s-graph, this partial scan circuit has no cycles. We will prove the optimality of this design by showing that no flip-flop in this

design can be dropped from scan without creating a cycle. Suppose we were to drop one flip-flop from scan. Because the s-graph is fully connected, the two non-scan flip-flops will form a cycle of length two. By a similar argument, no flip-flop can be dropped from scan without creating a cycle. Thus, the single non-scan flip-flop design is optimal. ■

14.11 Partial scan

By definition of a strongly connected component (SCC), every vertex in this graph lies on one or more cycles. A simple heuristic may select a vertex that is likely to be on the largest number of cycles. Deletion of such a vertex removes all those cycles. It has been observed[3] that a vertex with the highest product of indegree and outdegree offers a good choice. Once such a vertex and all its edges are deleted, the remaining s-graph may have one or more smaller SCCs. The same procedure of finding and deleting the vertex with the largest *indegree* × *outdegree* product is recursively applied until the remaining s-graph is free from SCCs.

14.12 Partial-scan overhead

In a partial-scan design no hardware or routing is added to non-scan flip-flops. So, for the calculation of the area overhead the non-scan flip-flop cells can be treated like combinational cells. In Equation 14.5, s is the cell area under flip-flops. We get the required area overhead simply upon replacing s by ps. Thus,

$$\begin{aligned} \text{Area overhead of partial} - \text{scan} &= \left[(1+\alpha ps)\left\{1+\frac{(1-\beta)k}{T}\right\}-1\right]\times 100\% \\ &\approx \left[\alpha ps+\frac{(1-\beta)k}{T}\right]\times 100\% \end{aligned}$$

14.13 Scan-hold flip-flops

The scan-hold flip-flop (SHFF) of Figure 14.13 (book, page 483) has four extra gates over the SFF of Figure 14.2. Since the SFF has four gates added already, the SHFF has eight more gates over a non-scan D flip-flop. Thus, the formula of Equation 14.2 (page 474) can be modified as,

$$\text{Gate overhead of SHFF design} = \frac{8\times n_{shff}}{n_g+10n_{ff}}\times 100\%$$

where n_{shff} is the number of SHFFs, n_{ff} is the number of flip-flops in the non-scan circuit, and n_g is the number of gates outside of flip-flops. Note that, in general, $n_{ff} \geq n_{shff}$, where the equality holds for a full-scan design.

When $n_g = 100,000$ and $n_{ff} = n_{shff} = 2,000$, the above formula gives an overhead of 13.3%, which is double that of the full-scan design with SFFs.

[3] S. Bhawmik, C. J. Lin, K.-T. Cheng and V. D. Agrawal, "PASCANT: A Partial Scan and Test Generation System," *Proc. IEEE Custom Integrated Circuits Conf.*, May 1991, pp. 17.3.1-17.3.4

14.14 Random-access scan

We use a word oriented memory to allow the simultaneous write and read of multiple bits. Since there are 10 output pins, we select a 10-bit word size. To have the capability of 1,000 flip-flops, we require a 100-word memory. There will then be $\lceil \log_2 100 \rceil = 7$ bits of address. One input pin will be used for the test control (TC) signal. The remaining 19 input pins will be reconfigured as 10 pins for input data, 7 pins for address, and one pin each for SEL and ACK signals shown in Figure 14.14 (page 485 of the book.) The 10 output pins will be multiplexed under the control of TC to the 10-bit memory output data.

Chapter 15: Built-In Self-Test

15.1 Test length

This solution is based on a paper by Wagner, *et al.*[4] If a fault is detected by k out of N vectors, then the probability of its first detection at vector t is given by the hypergeometric probability density function:

$$\text{Probability of first detection at vector } t = p_t = \frac{\begin{pmatrix} N-t \\ k-1 \end{pmatrix}}{\begin{pmatrix} N \\ k \end{pmatrix}}$$

The denominator is the number of ways in which k tests for the fault can possibly be distributed among N vectors. The numerator is the number of ways k tests can be arranged among N vectors such that (a) the first $t-1$ vectors do not detect the fault, (b) the t^{th} vector detects the fault, and (c) the remaining $t-1$ tests are randomly distributed among the remaining $N-t$ vectors. The average test length is given by,

$$T = \sum_{t=1}^{N} t p_t = \frac{N+1}{k+1}$$

where the manipulations leading to the above result may be found in the paper by Wagner, *et al.*

In the given case, $N = 15$ and $k = 2$. Thus, the average test length is,

$$T = \frac{15+1}{2+1} = \frac{16}{3} = 5.333$$

15.2 Standard LFSR

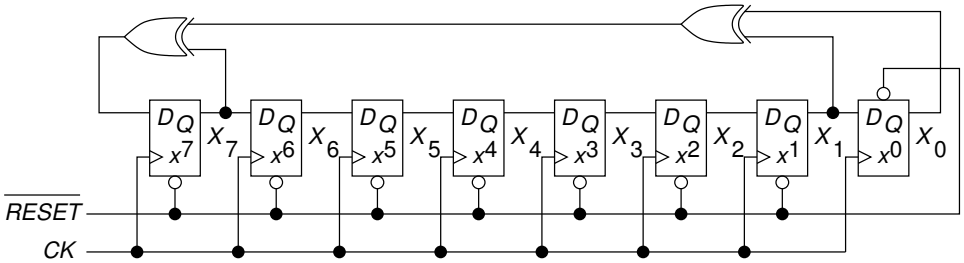
Consider the polynomial for a standard LFSR shown in the figure:

$$f(x) = x^8 + x^7 + x^2 + 1$$

$$\mathbf{X}(t+1) = \mathbf{T_s}\mathbf{X}(t)$$

$$\begin{bmatrix} X_0(t+1) \\ X_1(t+1) \\ X_2(t+1) \\ X_3(t+1) \\ X_4(t+1) \\ X_5(t+1) \\ X_6(t+1) \\ X_7(t+1) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_0(t) \\ X_1(t) \\ X_2(t) \\ X_3(t) \\ X_4(t) \\ X_5(t) \\ X_6(t) \\ X_7(t) \end{bmatrix}$$

[4] K. D. Wagner, C. K. Chin, and E. J. McCluckey, "Pseudorandom Testing," *IEEE Trans. on Computers*, vol. C-36, no. 3, pp. 332-343, March 1987.

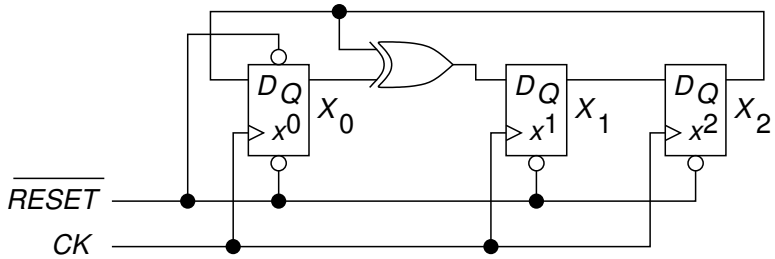


A standard LFSR

15.3 Modular LFSR

For the modular LFSR shown in the figure, consider the polynomial:

$$f(x) = x^3 + x + 1$$



Modular LFSR.

$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix}(t+1) = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix}(t)$$

15.4 Standard LFSR

Pattern #	X_7	X_6	X_5	X_4	X_3	X_2	X_1	X_0
1.	0	0	0	0	0	0	0	1
2.	1	0	0	0	0	0	0	0
3.	1	1	0	0	0	0	0	0
4.	1	1	1	0	0	0	0	0
5.	1	1	1	1	0	0	0	0
6.	1	1	1	1	1	0	0	0
7.	1	1	1	1	1	1	0	0
8.	0	1	1	1	1	1	1	0

15.5 Modular LFSR

Pattern #	X_0	X_1	X_2
1.	0	0	1
2.	1	1	0
3.	0	1	1
4.	1	1	1
5.	1	0	1
6.	1	0	0
7.	0	1	0
8.	0	0	1

15.6 MISRs

Equations representing MISR:

$$\mathbf{X}(t+1) = \mathbf{T_s}\mathbf{X}(t) + \mathbf{I}(t)$$

$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \\ X_7 \end{bmatrix}(t+1) = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \\ X_7 \end{bmatrix}(t) + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ B \\ A \end{bmatrix}$$

Equations for modular MISR:

$$\mathbf{X}(t+1) = \mathbf{T_s^T}\mathbf{X}(t) + \mathbf{I}(t)$$

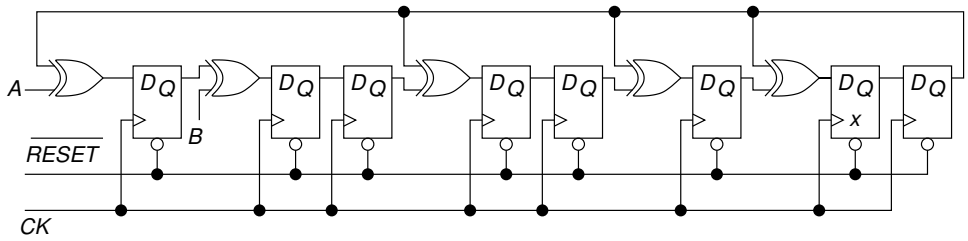
Standard equation: $\mathbf{X}(t+1) = \mathbf{T_s}\mathbf{X}(t)$

Transpose: $\mathbf{X^T}(t+1) = \mathbf{X^T}(t)\mathbf{T_s^T}$

Post-multiply both sides by $\mathbf{X^T}$ and pre-multiply both sides by $\mathbf{X_2}$, to get

$$\mathbf{X_2}(t+1) = \mathbf{T_s^T}\mathbf{X_2}(t)$$

$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \\ X_7 \end{bmatrix}(t+1) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \\ X_7 \end{bmatrix}(t) + \begin{bmatrix} A \\ B \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$



Taps: $h_3,\ h_5,\ h_6$

The modular LFSR gives the true remainder of the

$$\sum_{A,B}\left(\frac{\text{output sequence}}{\text{primitive polynomial}}\right)$$

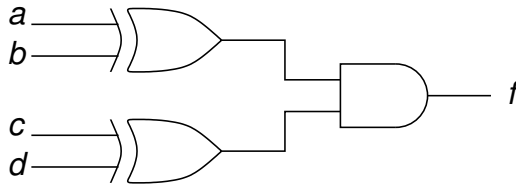
where $\sum$ is the XOR operator.

$$\begin{aligned} \mathbf{X}(t+1) &= \mathbf{T_s X}(t) \\ \mathbf{X^T}(t+1) &= (\mathbf{T_s X}(t))^{\mathbf{T}} \\ &= \mathbf{X^T}(t)\mathbf{T_s^T} \\ &= \mathbf{X^T}(t)\mathbf{T_M} \end{aligned}$$

The standard signature is a different state table realization of the modular MISR signature.

15.7 Weighted random patterns

The circuit under test is shown in the figure below and the following table gives ATPG generated patterns that provide 100% fault coverage.



100% *coverage vectors.*

Vector No.	Input (a, b, c, d)	Output f
1.	1110	0
2.	0101	1
3.	0010	0
4.	0110	1
5.	1000	0

The patterns generated by the circuit of Figure 15.16(b) (page 510 of the book) are given below:

Pattern #	X_7	X_6	X_5	X_4	X_3	X_2	X_1	X_0	1/2	1/4	1/8	1/16
1.	0	0	0	0	0	0	0	1	1	0	0	0
2.	1	0	0	0	0	0	0	0	0	0	0	0
3.	1	1	0	0	0	0	0	0	0	0	0	0
4.	1	1	1	0	0	0	0	0	0	0	0	0
5.	1	1	1	1	0	0	0	0	0	0	0	0
6.	1	1	1	1	1	0	0	0	0	0	0	0
7.	0	1	1	1	1	1	0	0	0	0	0	0
8.	0	0	1	1	1	1	1	0	0	0	0	0
9.	0	0	0	1	1	1	1	1	1	1	1	0
10.	1	0	0	0	1	1	1	1	1	1	0	0

We need two weight sets to test all faults:

A. Use bits $a = X_6$, $b = X_4$, $c = X_2$, $d = X_0$; this gets vectors 1, 4 and 5 of the 100% test set.

B. Replace X_4 with the 1/2 bit; this gets vectors 2, 3 and 5 of the test set.

Pattern #	Weight set A $abcd = X_6X_4X_2X_0$	Weight set B $abcd = X_6\frac{1}{2}X_2X_0$
1.	0001	**0101**
2.	0000	0000
3.	**1000**	**1000**
4.	**1000**	**1000**
5.	1100	**1000**
6.	1100	**1000**
7.	**1110**	1010
8.	**0110**	**0010**
9.	0111	0111
10.	0011	0111

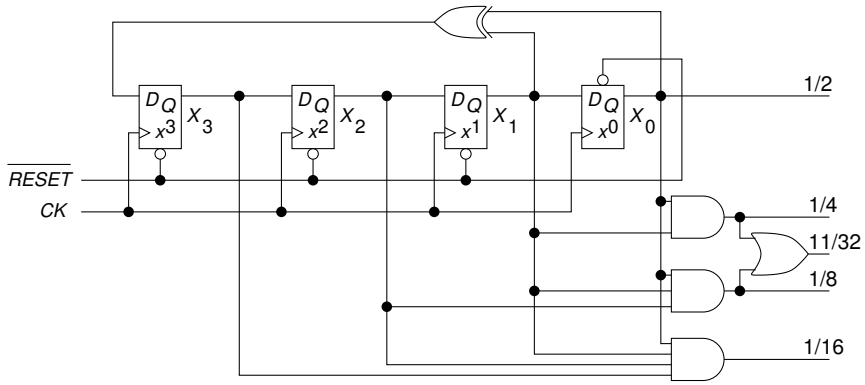
Required test vectors are shown in bold in the above table. Notice that none of the $\frac{1}{4}$, $\frac{1}{8}$ or $\frac{1}{16}$ bits are helpful here.

15.8 Weighted random pattern generator

Use a 4-bit pattern generator. From Appendix B of the book, the primitive polynomial is:

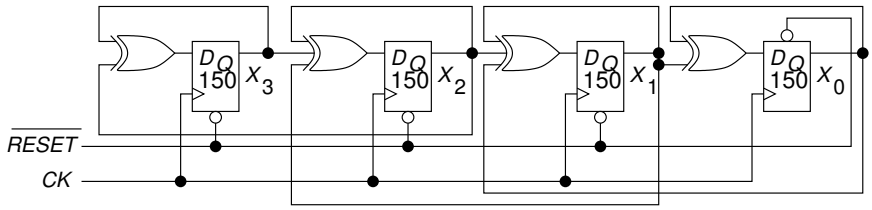
$$x^4 + x + 1$$

A circuit to generate the required weights is shown below.



15.9 Cellular automaton

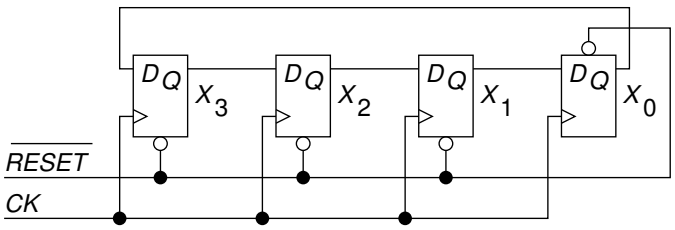
The CA pattern generator is shown below.



Starting with "0001" it generates patterns with a period of 6. These are as follows:

CA Pattern #	X_3	X_2	X_1	X_0
1.	0	0	0	1
2.	0	0	1	1
3.	0	1	0	0
4.	1	1	1	0
5.	0	1	0	1
6.	1	1	0	1
7.	0	0	0	1

The four flip-flop LFSR with non-primitive polynomial $1 + x^4$ and its patterns, starting from the initial pattern 0001, are shown next. Its period is 4.



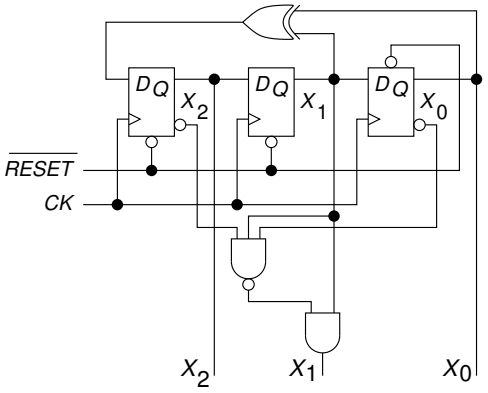
A non–primitive LFSR.

LFSR Pattern #	X_3	X_2	X_1	X_0
1.	0	0	0	1
2.	1	0	0	0
3.	0	1	0	0
4.	0	0	1	0
5.	0	0	0	1

The best system would be an LFSR with a primitive polynomial $f(x) = 1+x+x^4$, which would have a period of 15. For this example the CA is better than the non-primitive LFSR, because the CA has a longer period and is more random.

15.10 Maximal LFSR

A primitive polynomial (see Appendix B of the book) is $1 + x + x^3$. Using this polynomial we design the following maximal length (7) three-bit LFSR. The two-



gate circuit leading to the output X_1 converts 010 pattern to 000 without affecting all other patterns.

15.11 Aliasing probability

$p = 0.3$, $k = 15$,

$$p^k \leq P_{al} \leq (1-p)^k$$

$$1.435 \times 10^{-8} \leq P_{al} \leq 4.748 \times 10^{-3}$$

15.12 Fault detection

For multiple faults b s-a-0 and c s-a-0, we have

a	b	c	f_{good}	f_{bad}	S_{good}	S_{bad}
0	0	0	0	0	000	000
0	0	1	1	0	000	000
0	1	0	0	0	001	000
0	1	1	0	0	010	000
1	0	0	0	0	100	000
1	0	1	1	0	101	000
1	1	0	1	0	110	000
1	1	1	1	0	000	000
0	0	0	0	0	001	000

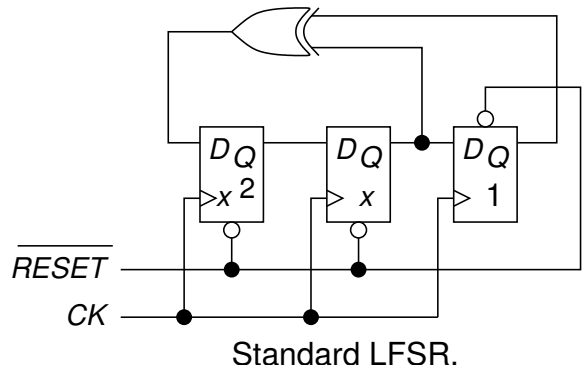
After 8 clocks:

Signature type	S_{good}	S_{bad}
LFSR	001	000
TC	4	0

Both the transition count (TC) and LFSR detect the multiple fault.

15.13 LFSR enhancement

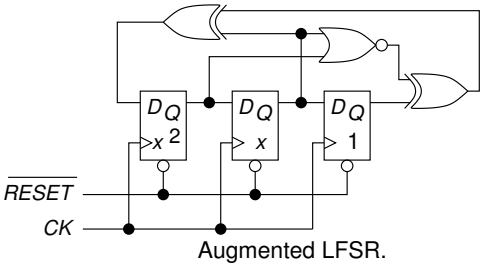
A standard LFSR and its patterns are shown below.



Standard LFSR.

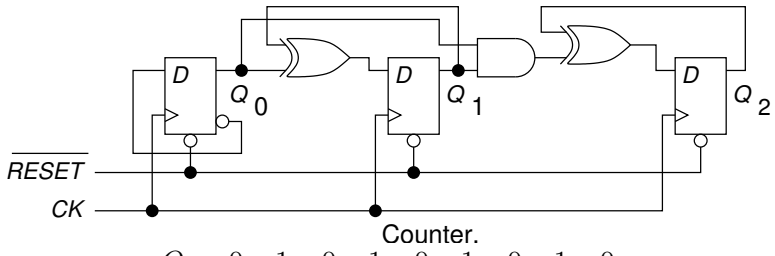
1	1	0	0	1	0	1	1	1
x	0	0	1	0	1	1	1	0
x^2	0	1	0	1	1	1	0	0

The next figure gives an augmented LFSR and the patterns it produces. This definitely uses less hardware than a counter, which needs more complex gates. It gets comparatively simpler as the counter width increases. A counter and its patterns are shown below.



Augmented LFSR.

1	0	0	0	1	0	1	1	1	0
x	0	0	1	0	1	1	1	0	0
x^2	0	1	0	1	1	1	0	0	0



Counter.

Q_0	0	1	0	1	0	1	0	1	0
Q_1	0	0	1	1	0	0	1	1	0
Q_2	0	0	0	0	1	1	1	1	0

15.14 Aliasing analysis

$$Z = Y(B \oplus C) \oplus B$$

Results of circuit simulation are as follows:

A	B	C	Y	Z	Good machine $R_1R_2R_3$	Failing machine, e sa0 $R_1R_2R_3$
0	0	1	D	D	000	000
1	0	0	D	0	011	000
0	1	0	D	$\overline{D}$	011	000
1	0	1	D	D	011	001
1	1	0	1	0	010	100
1	1	1	1	1	111	000
0	1	1	0	1	000	011
0	0	1	D	D	001	000
1	0	0	D	0	111	000

For output Y, the fault effect is XORed four times, while the fault effect is XORed into Z three times, during the first 7 clock periods. Repeating the first LFSR pattern during the 8th clock period XORs the fault effect in one additional time frame on each output.

The error vector is set to 1 on an output when it differs from a good machine. Here are the other error vectors:

	1	2	3	4	5	6	7	8
Y	1	1	1	1	0	0	0	1
Z	1	0	1	1	0	0	0	1

Even with the repeated pattern, the cumulative # of 1's in the error vector remains odd. This is why aliasing does not occur. If the total # of 1's in the error vector becomes even, then aliasing might occur.

15.15 Fault detection

ABC	Good	A s-a-0		A s-a-1		$B-e$ s-a-0	
	$R_1R_2R_3$	YZ	Bad $R_1R_2R_3$	YZ	Bad $R_1R_2R_3$	YZ	Bad $R_1R_2R_3$
001	000	11	000	11	000	11	000
100	011	10	011	10	011	10	011
010	011	10	011	10	011	10	011
101	011	11	011	11	011	11	011
110	010	10	010	10	010	10	010
111	111	01	111	11	111	11	111
011	000	01	010	11	000	10	000
001	001	11	100	11	011	11	010
	111		001 Yes		010 Yes		110 Yes

ABC	Good	$B-e$ s-a-1		$C-e$ s-a-0		$C-e$ s-a-1	
	$R_1R_2R_3$	YZ	Bad $R_1R_2R_3$	YZ	Bad $R_1R_2R_3$	YZ	Bad $R_1R_2R_3$
001	000	00	000	11	000	11	000
100	011	10	000	10	011	10	011
010	011	10	010	10	011	01	010
101	011	00	111	11	011	11	100
110	010	10	011	10	010	10	001
111	111	11	011	11	111	11	110
011	000	01	010	11	000	01	100
001	001	00	100	11	011	00	011
	111		010 Yes		010 Yes		001 Yes

15.16 Fault detection

ABC	Good	B s-a-0		B s-a-1		$B-g$ s-a-0	
	$R_1R_2R_3$	YZ	Bad $R_1R_2R_3$	YZ	Bad $R_1R_2R_3$	YZ	Bad $R_1R_2R_3$
001	000	11	000	01	000	11	000
100	011	10	011	10	001	10	011
010	011	10	011	10	110	11	011
101	011	11	011	11	101	11	010
110	010	10	010	10	101	11	110
111	111	11	111	11	100	10	100
011	000	11	000	01	001	01	000
001	001	11	011	01	101	11	001
	111		010 Yes		111 No		111 No

ABC	Good	$B-g$ s-a-1		f s-a-0		f s-a-1	
	$R_1R_2R_3$	YZ	Bad $R_1R_2R_3$	YZ	Bad $R_1R_2R_3$	YZ	Bad $R_1R_2R_3$
001	000	10	000	00	000	11	000
100	011	11	010	00	000	10	011
010	011	10	110	01	000	10	011
101	011	10	101	00	001	11	011
110	010	10	100	01	100	10	010
111	111	11	000	01	011	11	111
011	000	01	011	01	000	11	000
001	001	10	000	00	001	11	011
	111		010 Yes		100 Yes		010 Yes

15.17 Fault detection

ABC	Good	C s-a-0		C s-a-1		$C-g$ s-a-0	
	$R_1R_2R_3$	YZ	Bad $R_1R_2R_3$	YZ	Bad $R_1R_2R_3$	YZ	Bad $R_1R_2R_3$
001	000	10	000	11	000	10	000
100	011	10	010	11	011	10	010
010	011	10	111	01	010	10	111
101	011	10	001	11	100	10	001
110	010	10	110	11	001	10	110
111	111	10	101	11	111	10	101
011	000	10	100	01	000	01	100
001	001	10	000	11	001	10	011
	111		010 Yes		111 No		011 Yes

ABC	Good	$C-g$ s-a-1		$f-Y$ s-a-0		$f-Y$ s-a-1	
	$R_1R_2R_3$	YZ	Bad $R_1R_2R_3$	YZ	Bad $R_1R_2R_3$	YZ	Bad $R_1R_2R_3$
001	000	11	000	00	000	10	000
100	011	11	011	00	000	10	010
010	011	11	010	00	000	10	111
101	011	11	110	01	000	11	001
110	010	11	100	00	001	10	111
111	111	11	001	01	100	11	001
011	000	01	111	01	011	11	111
001	001	11	010	01	000	11	000
	111		110 Yes		001 Yes		011 Yes

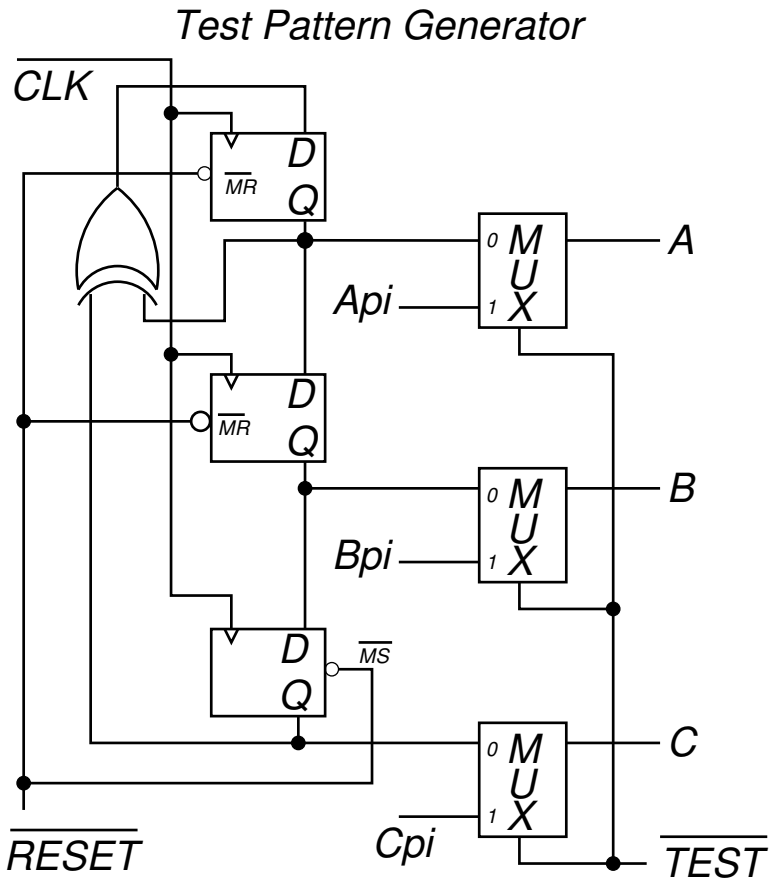
15.18 Fault detection

ABC	Good	$B-d$ s-a-0		$B-d$ s-a-1		$B-Z$ s-a-0	
	$R_1R_2R_3$	YZ	Bad $R_1R_2R_3$	YZ	Bad $R_1R_2R_3$	YZ	Bad $R_1R_2R_3$
001	000	11	000	11	000	11	000
100	011	10	011	10	011	10	011
010	011	10	011	10	011	11	011
101	011	11	011	11	011	11	010
110	010	10	010	10	010	11	110
111	111	01	111	11	111	10	100
011	000	01	010	01	000	10	000
001	001	11	100	11	001	11	010
	$\boxed{111}$		$\boxed{001}$ Yes		$\boxed{111}$ No		$\boxed{110}$ Yes

ABC	Good	$B-Z$ s-a-1		$f-k$ s-a-0		$f-k$ s-a-1	
	$R_1R_2R_3$	YZ	Bad $R_1R_2R_3$	YZ	Bad $R_1R_2R_3$	YZ	Bad $R_1R_2R_3$
001	000	10	000	10	000	11	000
100	011	11	010	10	010	10	011
010	011	10	110	11	111	10	011
101	011	10	101	10	000	11	011
110	010	10	100	11	010	10	010
111	111	11	000	11	110	11	111
011	000	01	011	01	100	01	000
001	001	10	000	10	011	11	001
	$\boxed{111}$		$\boxed{010}$ Yes		$\boxed{011}$ Yes		$\boxed{111}$ No

15.19 Signature computation

(a) The hardware is shown in following figure.



Circuit for Problem 15.19 with BIST pattern generator and input MUX.

(b)

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix}(t+1) = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix}(t)$$

(c) The table below contains the fault-free outputs of the circuit and the state of the MISR after every clock. The initial state of the flip-flops is assumed to be $Q_1Q_2Q_3 = 000$. The output equations used for computing the fault-free outputs in the table are:

$$Y = (A \oplus C) + AB \quad \text{and} \quad Z = \overline{B + C} \oplus Y$$

A	B	C	Y	Z	LFSR state		
					Q_1	Q_2	Q_3
0	0	0	0	1	0	0	1
0	0	1	1	1	1	1	1
0	1	0	0	0	0	1	1
0	1	1	1	1	0	1	0
1	0	0	1	0	1	1	1
1	0	1	0	0	0	1	1
1	1	0	1	1	0	1	0
1	1	1	1	1	1	1	0

Thus the final signature of the good machine is "1 1 0".

The logic to detect this signature can be implemented by a NAND gate as evident from the following equation.

$$\overline{GOOD} = \overline{Q_1 Q_2 \overline{Q_3}}$$

(d) In the case of the fault q s-a-0, the faulty outputs are:

$$Y_f = Y \quad \text{and} \quad Z_f = Y$$

The table below contains the faulty outputs of the circuit and the state of the LFSR after every clock. The initial state of the flip-flops is assumed to be $Q_1 Q_2 Q_3 = 000$ as before.

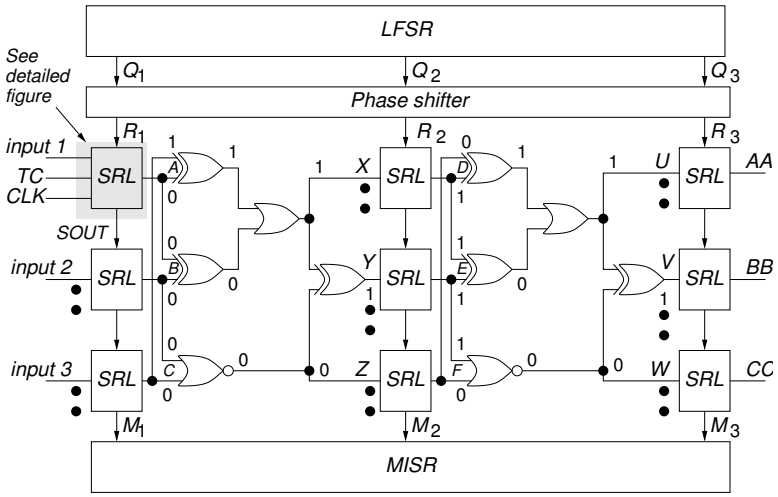
A	B	C	Y	Z	LFSR state		
					Q_1	Q_2	Q_3
0	0	0	0	0	0	0	0
0	0	1	1	1	0	1	1
0	1	0	0	0	0	0	1
0	1	1	1	1	1	1	1
1	0	0	1	1	0	0	0
1	0	1	0	0	0	0	0
1	1	0	1	1	0	1	1
1	1	1	1	1	0	1	0

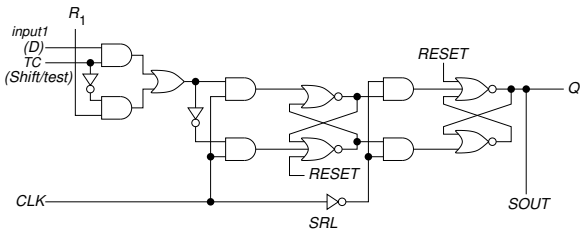
Thus, the final signature of the faulty circuit will be "0 1 0", and the test hardware does not alias.

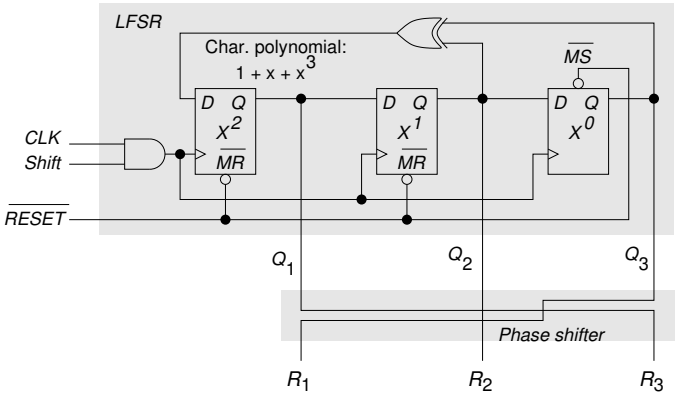
Solution provided by K. K. Saluja and M. L. Bushnell

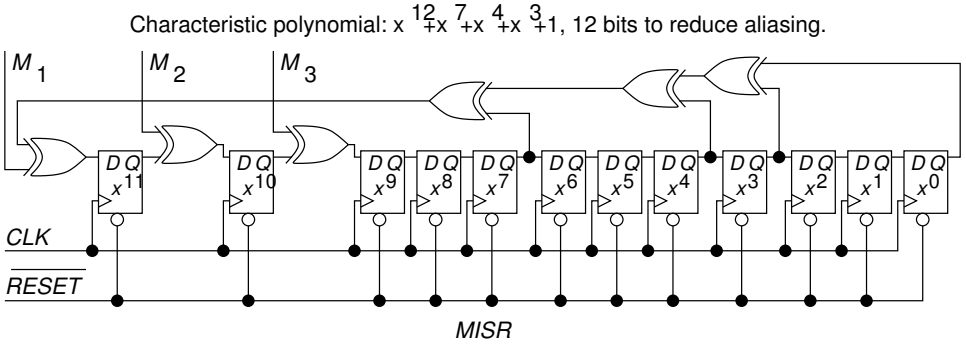
15.20 STUMPS

The four figures show the implementation of STUMPS, its basic component cell shift register latch (SRL), the LFSR and phase shifter, and the MISR. A logic design was synthesized using the Synopsys design compiler, which produced a netlist for simulation. The signature was obtained by simulation.










```
-- This is vhdl code describing the STUMPS BIST system.  The working
-- hardware is obtained from the Synopsys system.  Run the Synopsys
-- design_analyzer, read in this vhdl file, and call for high optimization
-- and boundary optimization in order to obtain a good logic design.
--
library ieee;
use ieee.std_logic_1164.all;

entity stumps is

    port (test: in std_logic;
          shift :  in std_logic;
          input1 : in std_logic;
          input2 : in std_logic;
          input3 : in std_logic;
          A, B, C, D, E, F: inout std_logic;
          AA : inout std_logic;
          BB : inout std_logic;
          CC : inout std_logic;
          clock : in std_logic;
          reset : in std_logic;
          X0, X1, X2, X3, X4, X5, X6, X7, X8, X9, X10, X11 : inout std_logic
);

end stumps;

architecture stumps_arch of stumps is

    signal Q1 : std_logic;
    signal Q2 : std_logic;
    signal Q3 : std_logic;
    signal D1 : std_logic;
    signal D2 : std_logic;
    signal D3 : std_logic;
    signal U, V, W, X, Y, Z : std_logic;

begin  -- stumps_arch
    X <= (A xor C) or (A xor B);
```

```
    Y <= X xor Z;
    Z <= B nor C;

    U <= (D xor F) or (D xor E);
    V <= U xor W;
    W <= E nor F;

    -- LFSR AND PHASE SHIFTER

lfsr : process (clock, reset, shift)

      begin  -- process lfsr
          -- activities triggered by asynchronous reset (active low)
          if reset = '0' then
              Q3 <= '1';
              Q2 <= '0';
              Q1 <= '0';
          -- activities triggered by rising edge of clock
          elsif (clock'event) and (clock = '1') then
             if (shift = '1') then
                Q3 <= D3;
                Q2 <= D2;
                Q1 <= D1;
             end if;
          end if;
      end process lfsr;

    D1 <= Q2 xor Q3;
    D2 <= Q1;
    D3 <= Q2;

misr: process (clock, reset, X0, X1, X2, X3, X4, X5, X6, X7,
               X8, X9, X10, X11)

      begin  -- process
          -- activities triggered by asynchronous reset (active low)
          if reset = '0' then
              X0 <= '0';
              X1 <= '0';
              X2 <= '0';
              X3 <= '0';
              X4 <= '0';
              X5 <= '0';
              X6 <= '0';
              X7 <= '0';
              X8 <= '0';
              X9 <= '0';
              X10 <= '0';
              X11 <= '0';
          -- activities triggered by rising edge of clock
```

```
        elsif clock'event and clock = '1' then
          -- MISR
            X0 <= X1;
            X1 <= X2;
            X2 <= X3;
            X3 <= X4;
            X4 <= X5;
            X5 <= X6;
            X6 <= X7;
            X7 <= X8;
            X8 <= X9;
            X9 <= CC xor X10;
            X10 <= F xor X11;
            X11 <= C xor (X7 xor X4 xor X3 xor X0);
        end if;
    end process;

  -- SRLS
srls: process (clock)

    begin  -- process
        -- activities triggered by asynchronous reset (active low)
        -- activities triggered by rising edge of clock
        if reset = '0' then
                A <= '0';
                B <= '0';
                C <= '0';
                D <= '0';
                E <= '0';
                F <= '0';
                AA <= '0';
                BB <= '0';
                CC <= '0';
        elsif clock'event and clock = '1' then
            if (shift = '1') then
                A <= Q3;
                B <= A;
                C <= B;
                D <= Q2;
                E <= D;
                F <= E;
                AA <= Q1;
                BB <= AA;
                CC <= BB;
            else
                if (test = '0') then
                    A <= input1;
                    B <= input2;
                    C <= input3;
                end if;
```

```
                D <= X;
                E <= Y;
                F <= Z;
                AA <= U;
                BB <= V;
                CC <= W;
            end if;
        end if;

    end process;

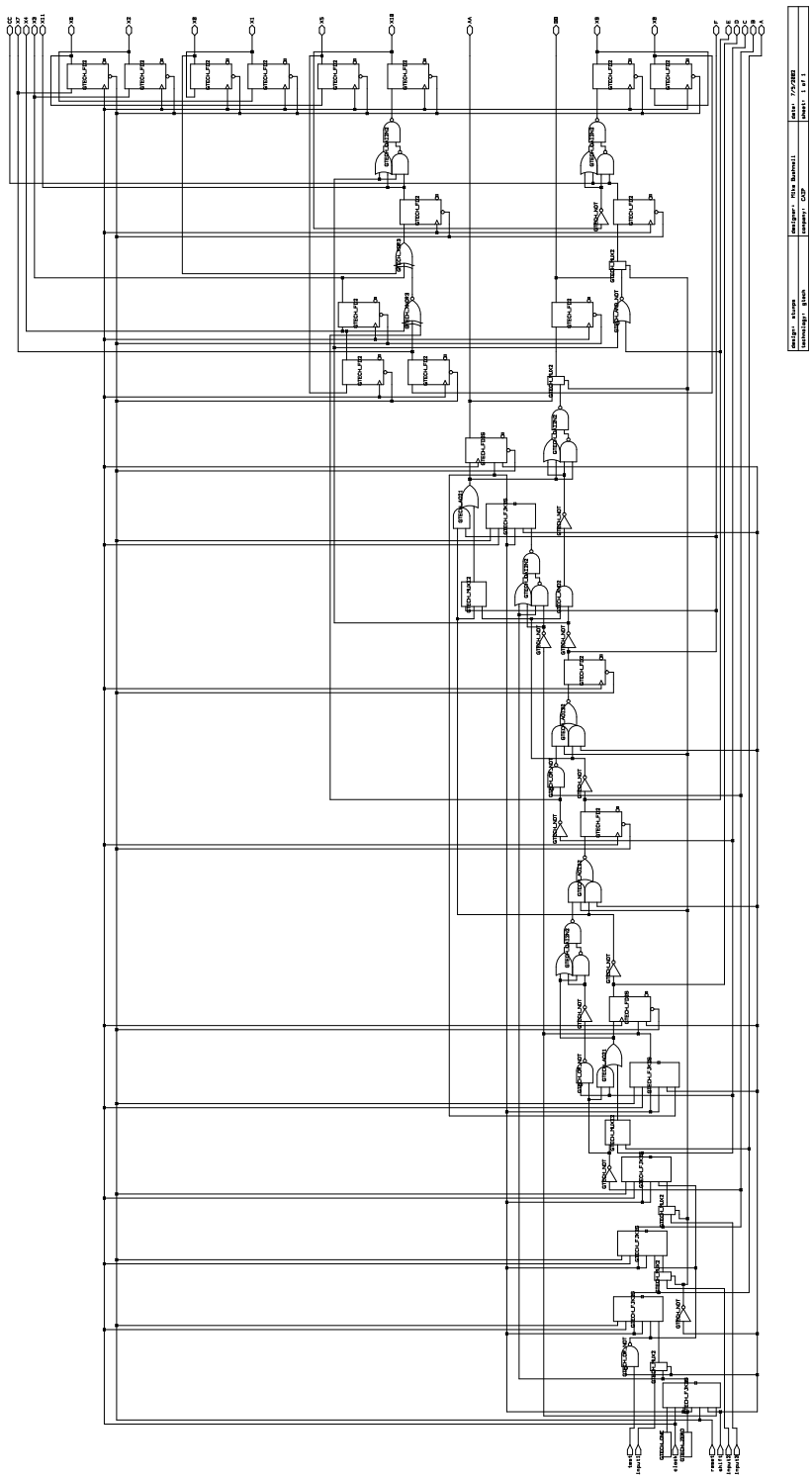
end stumps_arch;
```

A circuit diagram produced by Synopsys is shown on the next page. The final signature after 12 functional clock periods (each of which requires 3 more shifting clock periods) is:

$$(X11,\ \ldots,\ X0) = 9F0_{16}$$

The only practical way to compute this is with a simulator that would use the circuit description in a hardware description language such as VHDL or Verilog.

- Pros of STUMPS:
 1. Low hardware overhead (due to shift register latch structure and MISR that only has 1 position/scan chain.)
 2. Little DFT hardware is needed inside the circuit, except for the full-scan chains.
- Cons of STUMPS:
 1. This is a test-per-scan system. Each test pattern requires *scan chain length*+ 1 clocks. So, it is quite slow, and test time is long and costly.
- Observations:
 1. It was necessary to use 2 control pins, test and shift, for test mode, because in test mode, we still wanted to ignore the circuit inputs in the leftmost scan chain, whereas in the other scan chains, we wanted to capture the circuit responses in the scan chain.
 2. The VHDL simulation was very useful, as it caught conditions where X's were being clocked into the MISR.
 3. A 12-bit MISR was used to reduce aliasing.



15.21 MATS+ memory BIST

$$\text{MATS+} : \{M0 :\Updownarrow (w0);\ M1 :\Uparrow (r0, w1);\ M2 :\Downarrow (r1, w0)\}$$

```
-- This is vhdl code describing the memory BIST for MATS+.  The working
-- hardware is obtained from the Synopsys system.  Run the Synopsys
-- design_analyzer, read in this vhdl file, and call for high optimization
-- and boundary optimization in order to obtain a good logic design.
library ieee;
use ieee.std_logic_1164.all;

entity mats_plus is

  port (
    test : in std_logic;
    Last_address : in std_logic;
    First_address : in std_logic;
    clk: in std_logic;
    reset: in std_logic;
    data_out: in std_logic;
    CLEAR : out std_logic;
    up_address : out std_logic;
    COUNT : out std_logic;
    data_in : out std_logic;
    WRITE_CMD : out std_logic);

end mats_plus;

architecture mats_plus_arch of mats_plus is
  type states is (START, M0, M0s, M1r, M1w, M1rr, M1ww, ERRORS, M2r, M2w,
                  M2rr, M2ww, CORRECT);
  signal next_state, present_state: states;

  begin
    x: process (Last_address, First_address, test, data_out, present_state)
    begin  -- process

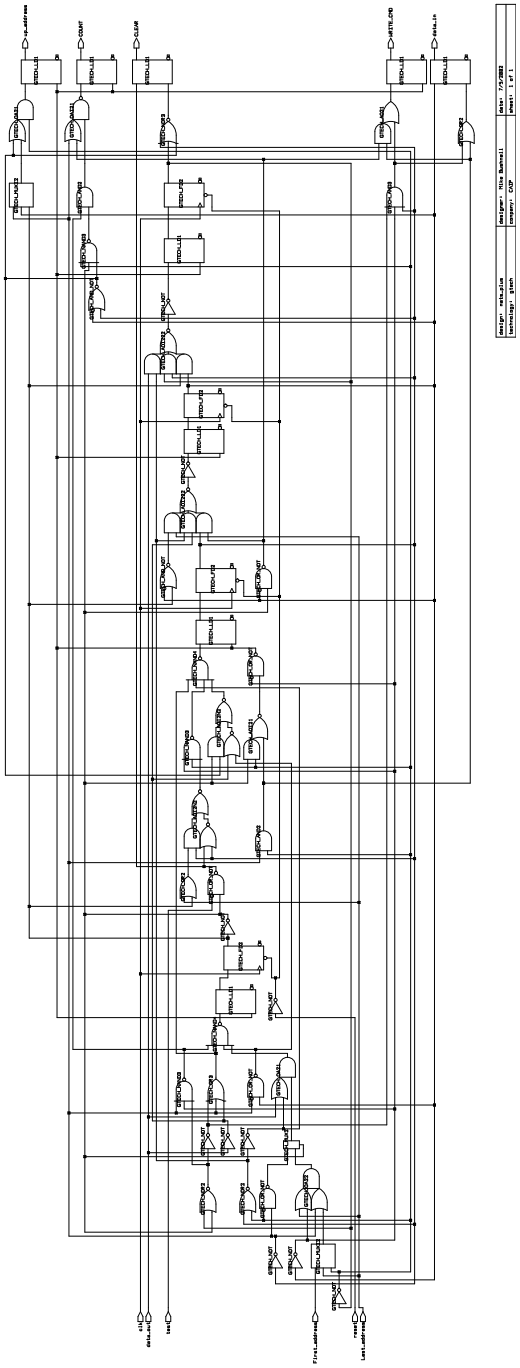
      case present_state is
        when START  =>
          up_address <= '0';
          COUNT <= '0';
          data_in <= '0';
          WRITE_CMD <= '0';
          if test = '0' then
            next_state <= START;
            CLEAR <= '0';
          else
            next_state <= M0;
            CLEAR <= '1';
```

```
  end if;
when M0  =>
  up_address <= '1';
  COUNT <= '1';
  data_in <= '0';
  WRITE_CMD <= '1';
  if Last_address = '0' then
    next_state <= M0;
  else
    next_state <= M0s;
  end if;
when M0s  =>
  up_address <= '1';
  COUNT <= '1';
  data_in <= '0';
  WRITE_CMD <= '1';
  CLEAR <= '1';
  next_state <= M1r;
when M1r  =>
  up_address <= '1';
  COUNT <= '0';
  WRITE_CMD <= '0';
  if data_out = '0' then
    next_state <= M1w;
  else
    next_state <= ERRORS;
  end if;
when M1w  =>
  up_address <= '1';
  COUNT <= '1';
  data_in <= '1';
  WRITE_CMD <= '1';
  if Last_address = '0' then
    next_state <= M1r;
  else
    next_state <= M1rr;
  end if;
when M1rr  =>
  up_address <= '1';
  COUNT <= '0';
  WRITE_CMD <= '0';
  if data_out = '0' then
    next_state <= M1ww;
  else
    next_state <= ERRORS;
  end if;
when M1ww  =>
  up_address <= '1';
  COUNT <= '0';
  data_in <= '1';
```

```
          WRITE_CMD <= '1';
          next_state <= M2r;
        when M2r  =>
          up_address <= '0';
          COUNT <= '0';
          WRITE_CMD <= '0';
          if data_out = '0' then
            next_state <= ERRORS;
          else
            next_state <= M2w;
          end if;
        when M2w  =>
          up_address <= '0';
          COUNT <= '1';
          data_in <= '0';
          WRITE_CMD <= '1';
          if First_address = '0' then
            next_state <= M2r;
          else
            next_state <= M2rr;
          end if;
        when M2rr  =>
          up_address <= '0';
          COUNT <= '0';
          WRITE_CMD <= '0';
          if data_out = '0' then
            next_state <= ERRORS;
          else
            next_state <= M2ww;
          end if;
        when M2ww  =>
          up_address <= '0';
          COUNT <= '0';
          data_in <= '0';
          WRITE_CMD <= '1';
          next_state <= CORRECT;
        when others => null;
      end case;
  end process;

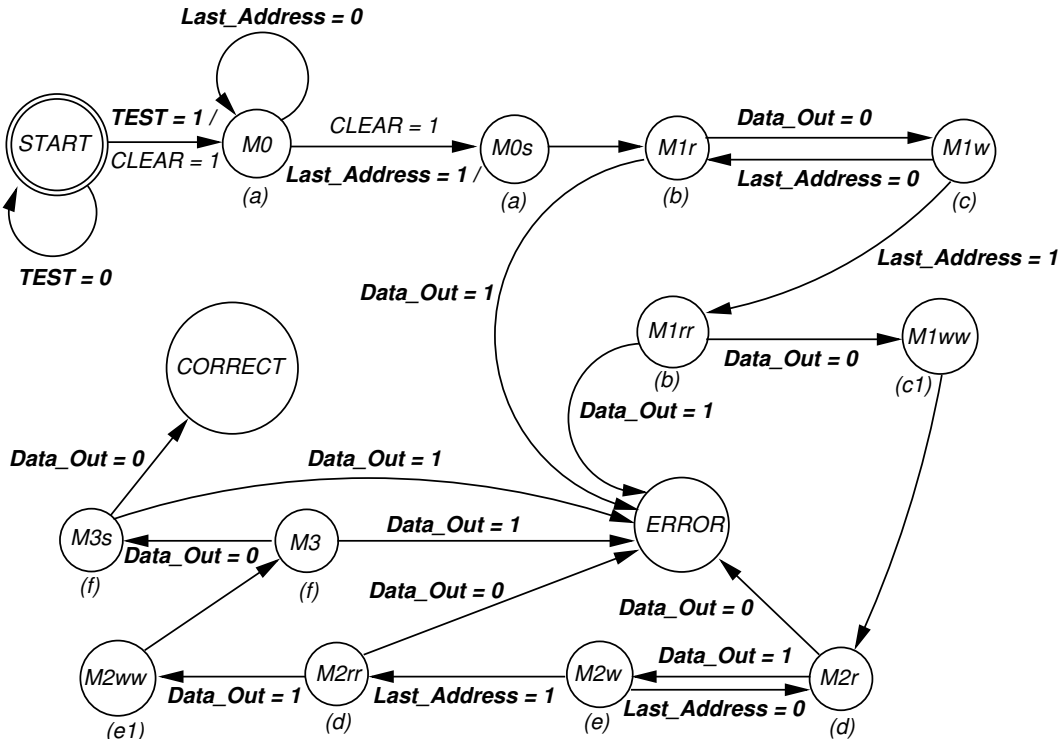
  ns: process (clk, test, reset)
  begin  -- process
    if reset = '1' then               -- asynchronous reset (active low)
      present_state <= START;
    elsif clk'event and clk = '1' then  -- rising clock edge
      present_state <= next_state;
    end if;
  end process;

end architecture mats_plus_arch;
```



15.22 MARCH X Memory BIST

$$\text{March X} \quad : \quad \{M0 :\Updownarrow (w0);\ M1 :\Uparrow (r0, w1);\ M2 :\Downarrow (r1, w0);\ M3 :\Updownarrow (r0)\}$$



Signals:

$TEST = 1$ (test mode), $TEST = 0$ (normal mode)
Up_Add (1 = count up, 0 = count down)
$COUNT$ (1 = step addresses, 0 = do not step)
$Data_In$ (data written to memory)
$Data_Out$ (data read from memory)
$WRITE$ (0 = read OP, 1 = write OP)
$Last_Add$ (1 means stepper just rolled over to last memory address)
$CLEAR$ (1 = clear address stepper to location 0)

(a): $Up_Add = 1$, $COUNT = 1$, $Data_In = 0$, $WRITE = 1$

(b): $Up_Add = 1$, $COUNT = 1$, $WRITE = 0$

(c): $Up_Add = 1$, $COUNT = 1$, $Data_In = 1$, $WRITE = 1$

(d): $Up_Add = 0$, $COUNT = 1$, $WRITE = 0$

(e): $Up_Add = 0$, $COUNT = 1$, $Data_In = 0$, $WRITE = 1$

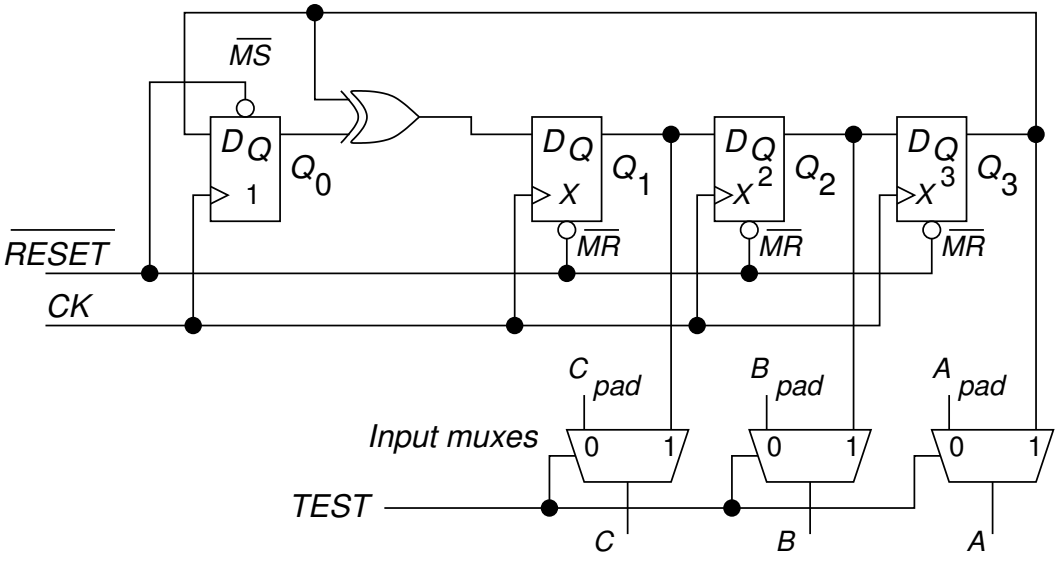
(f): $Up_Add = 1$, $COUNT = 1$, $WRITE = 0$

(c1): $Up_Add = 0$, $COUNT = 1$, $Data_In = 1$, $WRITE = 1$

(e1): $Up_Add = 0$, $COUNT = 0$, $CLEAR = 1$, $Data_In = 0$, $WRITE = 1$

15.23 BIST system

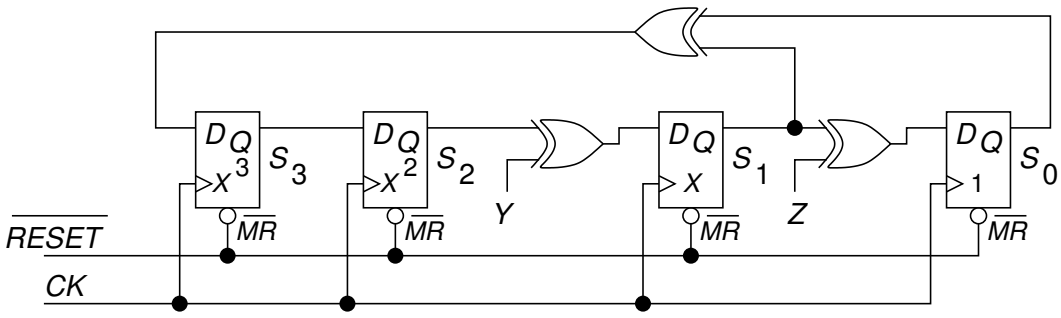
(a) The modular LFSR circuit for the CUT of Figure 15.54 (page 548 of the book) is shown below. The multiplexers disconnect the normal inputs A_{pad}, B_{pad} and C_{pad}, and feed the LFSR-generated patterns to the inputs A, B and C of the CUT.



(b)

$$\begin{bmatrix} Q_0(t+1) \\ Q_1(t+1) \\ Q_2(t+1) \\ Q_3(t+1) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} Q_0(t) \\ Q_1(t) \\ Q_2(t) \\ Q_3(t) \end{bmatrix}$$

(c) Probability of aliasing in the 4-bit MISR $= 1/2^4 = 1/16$, whereas for a 2-bit MISR it is $1/2^2 = 1/4$. The next figure shows the MISR circuit which receives the PO signals X and Y from the CUT of Figure 15.54 (page 648 of the book.)



(d)

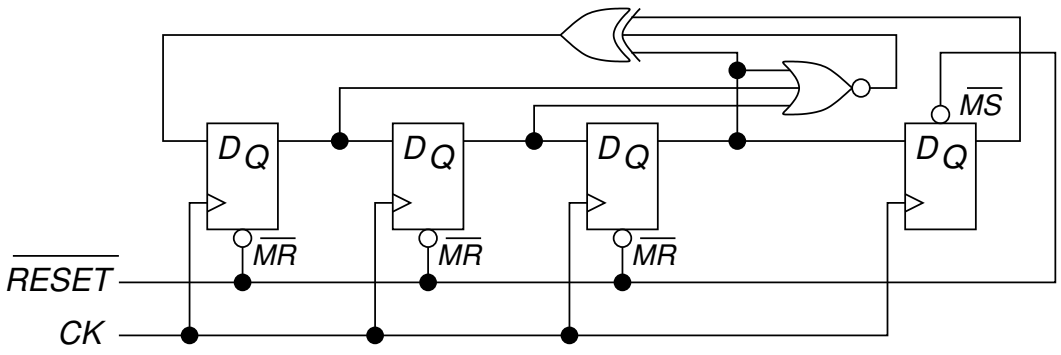
$$\begin{bmatrix} S_0(t+1) \\ S_1(t+1) \\ S_2(t+1) \\ S_3(t+1) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} S_0(t) \\ S_1(t) \\ S_2(t) \\ S_3(t) \end{bmatrix} + \begin{bmatrix} Z \\ Y \\ 0 \\ 0 \end{bmatrix}$$

(e) The circuit has an internal flip-flop, driven by n and driving signal o. There is no initialization hardware on this flip-flop, so it comes up in a random state. The variation in signatures happens because sometimes the flip-flop initializes as 0, and sometimes as 1.

We fix this problem bu adding a $\overline{RESET}$ line to this flip-flop and initializing it to 0 when we initialize the LFSR and MISR.

15.24 Up/Down LFSR

From Appendix B, a primitive polynomial for a 4-bit LFSR is $f(x) = 1+x+x^4$. The following circuit produces all patterns including 0000, which appears immediately after the initialization pattern, 0001. A NOR gate has been added to the basic LFSR to produce the 0000 pattern. The pattern sequence is shown after the circuit diagram.



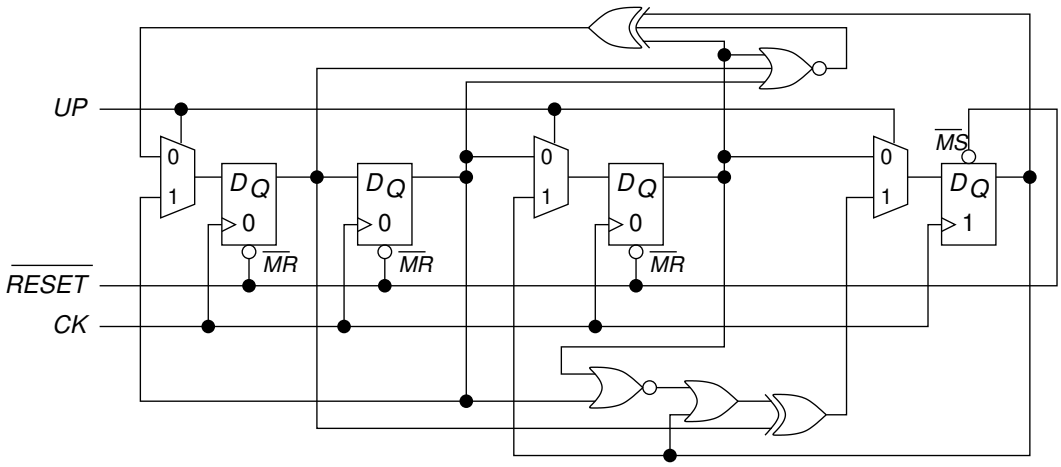
Pattern No.	Pattern	Decimal value	Remarks
1	0001	1	Initialization pattern
2	0000	0	Forced by NOR gate
3	1000	8	
4	0100	4	
2	0010	2	
6	1001	9	
7	1100	12	
8	0110	6	
9	1011	11	
10	0101	5	
11	1010	10	
12	1101	13	
13	1110	14	
14	1111	15	
15	0111	7	
16	0011	3	
17	0001	1	Sequence starts repeating

To find the inverse LFSR, we compute the inverse characteristic function:

$$\frac{1 + x + x^2 + x^3 + x^4 + x^5 + x^6 + x^7 + x^8}{f(x)}$$

$$\begin{array}{rcccccccccc}
 & 1 & & x^2 & & x^4 & & & & & \\
\cline{2-10}
1 + x + x^4 \;\big| & 1 & +x & +x^2 & +x^3 & +x^4 & +x^5 & +x^6 & +x^7 & +x^8 & \\
 & 1 & +x & & & +x^4 & & & & & \\
\cline{2-10}
 & & & x^2 & +x^3 & & +x^5 & +x^6 & +x^7 & +x^8 & \\
 & & & x^2 & +x^3 & & & +x^6 & & & \\
\cline{3-10}
 & & & & & & x^5 & & +x^7 & +x^8 & \\
 & & & & & x^4 & +x^5 & & & +x^8 & \\
\cline{5-10}
 & & & & & x^4 & & & +x^7 & &
\end{array}$$

This does not evenly divide the all 1's polynomial and we get a remainder of $x^4 + x^7$. We conclude that the inverse LFSR does not exist, so we must synthesize it as a finite state machine. The following circuit is based on a design synthesized by Synopsys.

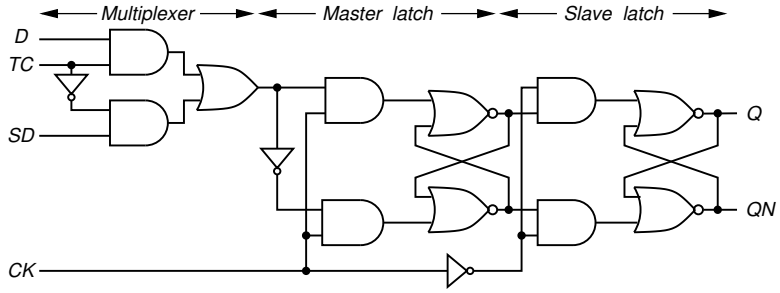


Up/Down LFSR of Problem 15.24

Chapter 16: Boundary Scan Standard

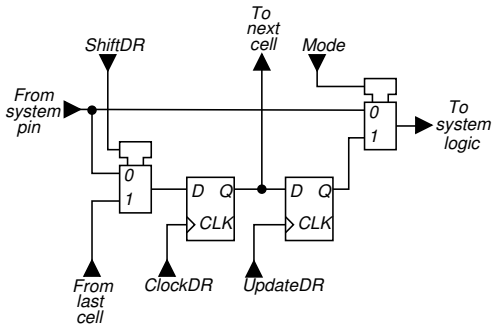
16.1 Boundary scan design for testability

(a) A full-scan flip-flop is shown in the figure below.



$$\text{Overhead of full} - \text{scan} = \left(\frac{g + 14f}{g + 10f}\right) \times 100\%$$

(b) A JTAG boundary scan cell is shown in the figure below.



A MUX has four gates and a flip-flop has ten gates. Therefore,

$$\text{Overhead of boundary scan} = \left(\frac{g + 14f + 28(pi + po)}{g + 14f}\right) \times 100\%$$

16.2 Boundary scan economics

Boundary scan uses two MUXes and two FF's per pin. With four gates (14 transistors) per MUX and ten gates (44 transistors) per master-slave FF, we get

$$\text{Transistors per I/O} = 2 \times 14 + 2 \times 44 = 116$$

$$\begin{aligned} \text{Hardware cost} &= \#pins \times \frac{\#transistors}{\#\ of\ I/O} \times \frac{cost}{transistor} + TAP \\ &= (256 \times 116 + 262) \times 525 \times 10^{-6} \text{ cents} = 15.73 \text{ cents} \end{aligned}$$

$$\begin{aligned}
\text{Test time in TCK's} &= 5 + \#pins \times \#vectors \\
\text{Test time cost} &= \frac{5 + 256 \times 512,000}{200 \times 10^6\ Hz} \times (4.5 \text{ cents/s}) \\
&= 2.94912 \text{ cents}
\end{aligned}$$

16.3 Boundary scan test time

Eack chip has a boundary scan register. It takes 5 clocks to go from shift-DR through update-DR and back to shift-DR.

First scenario:

$$\begin{aligned}
\text{Pattern shift} - \text{in time} &= \text{shift} - \text{out time} \\
&= 1024 + 512 + 1024 + 512 = 3072
\end{aligned}$$

$$\begin{aligned}
\text{Total test time} &= [\text{Chip 1 time} + \text{Chip 2 time} \\
&\quad \text{Chip 3 time} + \text{Chip 4 time}]/\text{clock} \\
&= [500,000(\text{shift time} + 5) + \text{shift time} \\
&\quad +1,000,000(\text{shift time} + 5) + \text{shift time} \\
&\quad +1,000,000(\text{shift time} + 5) + \text{shift time} \\
&\quad +500,000(\text{shift time} + 5) + \text{shift time}]/\text{clock}
\end{aligned}$$

However, three of the extra shift times can be eliminated by overlap of scan out with scan in as testing shifts from one chip to the next one.

$$\begin{aligned}
\text{Total test time} = &= [3,000,000 \times 3,077 + 3,072]/512 \text{ MHz} \\
&= \boxed{18.0293 \text{ s}}
\end{aligned}$$

Second scenario with bypass register:

$$\begin{aligned}
\text{Chip 1 shift time} &= 1027 \\
\text{Chip 2 shift time} &= 515 \\
\text{Chip 3 shift time} &= 1027 \\
\text{Chip 4 shift time} &= 515 \\
\text{Total test time} &= [500,000(1027 + 5) + 1027 \\
&\quad +1,000,000(515 + 5) + 515 \\
&\quad +1,000,000(1027 + 5) + 1027 \\
&\quad +500,000(515 + 5) + 515]/512 \text{ MHz} \\
&= \boxed{4.54688 \text{ s}}
\end{aligned}$$

To be really thorough, we should add the time to change test instructions. Let us assume a 6-bit instruction register. Then,

$$\begin{aligned}\text{Test instruction change time} &= (3 + 24 \times 1 + 3)/\text{clock} \\ &\quad \text{(See Figure 16.11 TAP controller state diagram)} \\ &= 30/(512\text{ MHz}) = \boxed{5.859375{\times}10^{-8}\text{ s}}\end{aligned}$$

Updated times with instruction changes:

First scenario:

$$\begin{aligned}\text{Test time} &= \text{Prior test time} + 1\text{ instruction change} \\ &= 18.0293 + 5.859375 \times 10^{-8}\text{ s} = 18.0293\text{ s}\end{aligned}$$

Second scenario:

$$\begin{aligned}\text{Test time} &= \text{Prior test time} + 4\text{ instruction changes} \\ &= 4.54688 + 5.859375 \times 10^{-8}\text{ s} = 4.54688\text{ s}\end{aligned}$$

So, test instruction changing time is negligible for large chips.

16.4 Interconnect test time

$$\begin{aligned}\#\text{tests} &= 2 \times \#\text{outputs} = \frac{2(1024 + 512 + 1024 + 512)}{2} \\ &= 3,072\end{aligned}$$

It takes five clocks to go from shift-DR through update-DR and back to shift-DR.

$$\begin{aligned}\text{Test time} &= \frac{3,072(\#\text{shift clocks} + 5) + 1\text{ shift}}{\text{clock}} \\ &= \frac{3072[1024 + 512 + 1024 + 512 + 5] + 3072}{100\text{ MHz}} \\ &= 0.09455616\text{ s}\end{aligned}$$

We cannot use *BYPASS* mode, because all chips receiving a signal of an interconnect under test must also have their pins sampled during test of that interconnect. So, all boundary registers of all chips must be active throughout interconnect test. Also, correct board input signals must be applied to the board inputs during the interconnect test.

Sequence of JTAG commands:

1. Apply the *PRELOAD* instruction with 3072 clocks to set the hold registers to a known state.

2. Apply the *EXTEST* instruction.

3. Apply enough *TCK* signals to shift the next interconnect test pattern (and shift out the results of the last interconnect test that were captured in all boundary registers.) Apply the necessary printed circuit board (PCB) input test vector to all PCB inputs. Then, apply a functional clock.

4. Loop back to step 2 if there are more interconnect tests to apply. Otherwise, just apply enough *TCK* signals to shift out the last response to the last interconnect test from the boundary scan register.

16.5 Interconnect delay test time

$$\begin{aligned} \#\text{tests} &= 2 \times \#\text{outputs} = \frac{2(500,000 + 1,000,000 + 1,000,000)}{2} \\ &= 3,072 \end{aligned}$$

It takes five clocks to go from *Shift-DR* through *Update-DR* and back to *Shift-DR*. Each test requires two patterns to shift in.

$$\begin{aligned} \text{Test time} &= \frac{\text{preload} + \#\text{tests} \times \frac{\#\text{shifts}}{\text{test}} + 1 \text{ shift}}{\text{clock}} \\ &= \frac{3072 + 5 + 3072(3072 + 5 + 3072 + 5) + 3072}{100 \text{ MHz}} \\ &= 0.18911237 \text{ s} \end{aligned}$$

Sequence of JTAG commands:

1. Apply the *PRELOAD* instruction with 3072 clocks to set the hold register to a known state.

2. Apply the *EXTEST* instruction.

3. Apply enough *TCK* signals (3072) to shift in the next interconnect test pattern (first pattern.) Apply the necessary printed circuit board (PCB) input test vector to all PCB inputs. Then, apply a functional clock. This also shifts out the response to the last interconnect test.

4. Apply enough *TCK* signals (3072) to shift in the second pattern for the interconnect test. Apply the necessary PCB input test vector to all PCB inputs. Then, apply a functional clock.

5. Loop back to step 2 if there are more interconnect tests to apply. Otherwise, apply 3072 *TCK*'s to shift out the response to the very last interconnect test from the boundary scan register.

16.6 Boundary scan delay fault test

Sequence of JTAG commands used to test the path:

1. *SAMPLE* – Capture pin signals and functional hardware outputs in boundary scan register.

2. Apply *INTEST* – Copy boundary scan register contents to hold latch. Apply as many *TCK* pulses as are necessary to shift the first test pattern in from *TDI* (this will be the number of pins from *TDI* up to and including this system input pin.) Then apply a functional system clock.

3. Repeat step 2 to apply the first time frame pattern to the path and shift the second time frame pattern into the boundary scan register (requires many *TCK* pulses.) At this point, the first time frame pattern is applied from the hold registers and the second time frame pattern is in the boundary scan register.

4. Repeat the *INTEST* instruction and immediately apply a functional system clock. This applies the second time frame pattern to the path under test, clocks the functional circuit, and captures its response in the boundary scan register. Then, apply as many *TCK* pulses as necessary to shift out the response to the second pattern out through *TDO* (while simultaneously shifting in the next test pattern.) The shifted out response will indicate where a timing fault has occurred.

5. Go back to step 2 if more paths are to be tested.

This sequence can be preceded by *IDCODE* and *USERCODE* instructions, in order to identify which chip, and which part programmings, are in the system. Other chips in the system may be given *BYPASS* instructions to speed up scan chain shifting.

16.7 Test controller

The following VHDL code describes the test controller of Figure 16.11 (page 558 of the book.) It is used to synthesize a circuit by Synopsys. The results of ATPG and an schematic of the synthesized circuit follow the VHDL code.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity TAP is port (TCK:     in std_logic;
                    TMS:     in std_logic;
                    TRST:    in std_logic;
                    outputs: out std_logic_vector (0 to 3));
end TAP;
```

```
architecture TAP_imp of TAP is
  constant Test_Logic_Reset: std_logic_vector (0 to 3) := X"0";
  constant Run_Test_Idle:    std_logic_vector (0 to 3) := X"1";
  constant Select_DR_Scan:   std_logic_vector (0 to 3) := X"2";
  constant Capture_DR:       std_logic_vector (0 to 3) := X"3";
  constant Shift_DR:         std_logic_vector (0 to 3) := X"4";
  constant Exit1_DR:         std_logic_vector (0 to 3) := X"5";
  constant Pause_DR:         std_logic_vector (0 to 3) := X"6";
  constant Exit2_DR:         std_logic_vector (0 to 3) := X"7";
  constant Update_DR:        std_logic_vector (0 to 3) := X"8";
  constant Select_IR_Scan:   std_logic_vector (0 to 3) := X"9";
  constant Capture_IR:       std_logic_vector (0 to 3) := X"A";
  constant Shift_IR:         std_logic_vector (0 to 3) := X"B";
  constant Exit1_IR:         std_logic_vector (0 to 3) := X"C";
  constant Pause_IR:         std_logic_vector (0 to 3) := X"D";
  constant Exit2_IR:         std_logic_vector (0 to 3) := X"E";
  constant Update_IR:        std_logic_vector (0 to 3) := X"F";
  signal next_state: std_logic_vector (0 to 3);
  signal present_state: std_logic_vector (0 to 3);

begin  -- TAP_imp

outputs <= present_state;

next_st : process (present_state, TMS)

begin  -- process next_st
    case present_state is
        when Test_Logic_Reset =>
          if TMS = '0' then
            next_state <= Run_Test_Idle;
          else
            next_state <= Test_Logic_Reset;
          end if;
        when Run_Test_Idle =>
          if TMS = '0' then
            next_state <= Run_Test_Idle;
          else
            next_state <= Select_DR_Scan;
          end if;
        when Select_DR_Scan =>
          if TMS = '0' then
            next_state <= Capture_DR;
          else
            next_state <= Select_IR_Scan;
          end if;
        when Capture_DR =>
          if TMS = '0' then
            next_state <= Shift_DR;
          else
```

```
    next_state <= Exit1_DR;
  end if;
when Shift_DR   =>
  if TMS = '0' then
    next_state <= Shift_DR;
  else
    next_state <= Exit1_DR;
  end if;
when Exit1_DR   =>
  if TMS = '0' then
    next_state <= Pause_DR;
  else
    next_state <= Update_DR;
  end if;
when Pause_DR  =>
  if TMS = '0' then
    next_state <= Pause_DR;
  else
    next_state <= Exit2_DR;
  end if;
when Exit2_DR   =>
  if TMS = '0' then
    next_state <= Shift_DR;
  else
    next_state <= Update_DR;
  end if;
when Update_DR =>
  if TMS = '0' then
    next_state <= Run_Test_Idle;
  else
    next_state <= Select_DR_Scan;
  end if;
when Select_IR_Scan =>
  if TMS = '0' then
    next_state <= Capture_IR;
  else
    next_state <= Test_Logic_Reset;
  end if;
when Capture_IR =>
  if TMS = '0' then
    next_state <= Shift_IR;
  else
    next_state <= Exit1_IR;
  end if;
when Shift_IR  =>
  if TMS = '0' then
    next_state <= Shift_IR;
  else
    next_state <= Exit1_IR;
  end if;
```

```
        when Exit1_IR   =>
          if TMS = '0' then
            next_state <= Pause_IR;
          else
            next_state <= Update_IR;
          end if;
        when Pause_IR   =>
          if TMS = '0' then
            next_state <= Pause_IR;
          else
            next_state <= Exit2_IR;
          end if;
        when Exit2_IR =>
          if TMS = '0' then
            next_state <= Shift_IR;
          else
            next_state <= Update_IR;
          end if;
        when Update_IR =>
          if TMS = '0' then
            next_state <= Run_Test_Idle;
          else
            next_state <= Select_DR_Scan;
          end if;
        when others     => next_state <= Test_Logic_Reset;
    end case;
end process next_st;

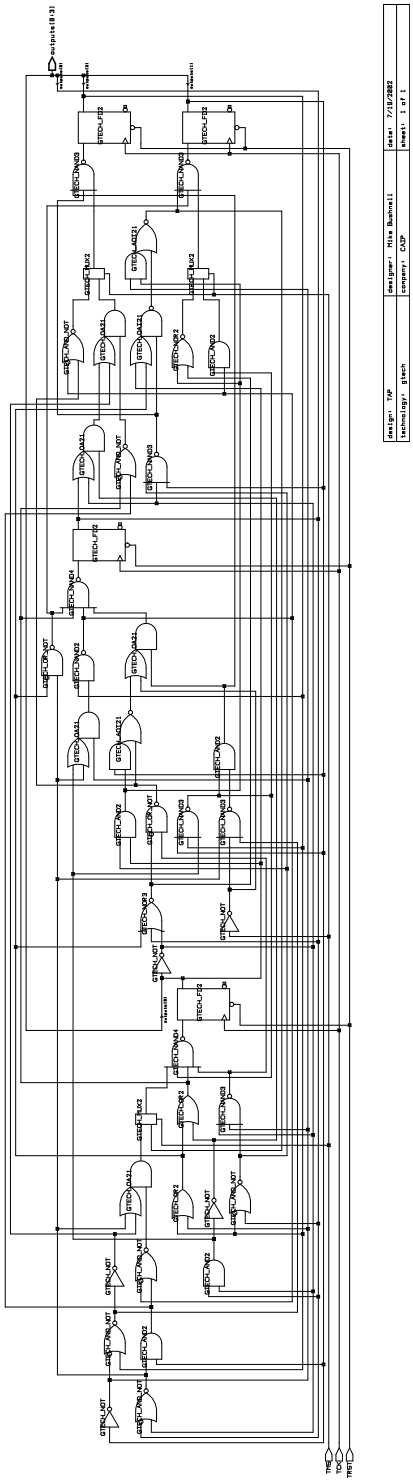
outputs <= present_state;

clocking: process (TCK, TRST)
begin
    if TRST = '0' then
        present_state <= Test_Logic_Reset;
    elsif (TCK'event and TCK = '1') then
        present_state <= next_state;
    end if;
end process clocking;

end TAP_imp;
```

The synthesized controller circuit (see next page) contains three PIs, four POs, about 60 gates and four flip-flops. A sequential circuit ATPG, available in the Synopsys Design Analyzer, was used to generate tests for this circuit:

- Total number of collapsed stuck-at faults = 332
- Number of vectors = 28
- Fault coverage = 100%



16.8 Bus and controller testing

Let i = # bits in a JTAG instruction and b = # boundary register cells for System Logic 2 chip. Test for System Logic 1 chip and interconnect:

1. Reset *TAP* with *TRST** and apply 1 test clock.
$$1\ TCK$$

2. Set System Logic 1 chip in *PRELOAD* mode and System Logic 2 chip in *PRELOAD* mode. Shift in pattern that stores '1' in 3 *CS* lines driven by System Logic 1 Chip and 1 in bus enable signal driven by System Logic 2 Chip.
$$6 + 2i + 5 + 6 + b\ TCK's$$

3. Set System Logic 1 chip in *INTEST* mode and System Logic 2 chip in *BYPASS* mode.
$$6 + 2i\ TCK's$$

4. For each test pattern for System Logic 1 chip (512):

 a. Scan in the test vector and scan out prior response.
 $$2 + 7\ TCK's$$

 b. Functionally clock the system to apply the vector and capture the response.
 $$3\ TCK's$$

5. Scan out the last vector response.
$$2 + 7 + 3\ TCK's$$

6. For each of 3 memory chips:

 a. Put System Logic 1 and 2 chips in *EXTEST* mode.
 $$6 + 2i\ TCK's$$

 b. Shift in a vector that sets the *CS* output of System Logic 1 chip for the memory chip to 0 and that sets the *CS* outputs for the other two memory chips to 1. The vector should also set the outputs of System Logic 2 chip as follows:

 Bus address = 0

 Data = 0

 Command = *write*

 Bus enable = 1
 $$2 + 6 + b + 3\ TCK's$$

 c. Shift in a vector that sets the *CS* output of System Logic 1 chip for the memory chip to 0 and that sets the *CS* outputs for the other two memory chips to 1. The vector should also set the outputs of System Logic 2 chip as follows:

 Bus address = 0

Data = 0
Command = *read*
Bus enable = 0

$$2 + 6 + b + 3 \; TCK's$$

These two steps check the *CS* line and verify successful writing and read back of 0 from System Logic 2 chip to the memory chip.

d. Repeat steps 6a and 6b, but with data set to a 1 to test writing and reading of 1.

$$2 \times (2 + 6 + b + 3) \; TCK's$$

e. Repeat steps 6a, 6b and 6c, but with address set to all 1's to fully test the address lines.

$$4 \times (2 + 6 + b + 3) \; TCK's$$

$$\begin{aligned} \text{Total test time} &= 1 + (6 + 2i + 5 + 6 + b) + 6 + 2i \\ &\quad +512(9 + 3) + 2 + 7 + 3 + 3[6 + 2i + 8(2 + 6 + b + 3)] \\ &= 10i + 25b + 6462 \; TCK's + 512 \text{ fuctional clocks} \end{aligned}$$

16.9 Memory testing

March C− has 10 March elements. We assume the memory chip loads a 4-bit word at a time and reads out a 4-bit word at a time.

$$\#\text{test patterns} = 10 \times 10^{28} = 2684354560$$

There is a clock overhead of 5 test clocks to apply a pattern after shifting it in. So,

$$\begin{aligned} \text{Shift time} &= 64 \\ \text{Total test time} &= \frac{\#\text{patterns}(\text{shift time} + 1 + 5) + \text{shift time}}{\text{clock rate}} \\ &= \frac{2684354560 \times 70 + 64}{200 \text{ MHz}} \\ &= \boxed{939.53 \text{ s}} \end{aligned}$$

Memory testing (alternative solution)

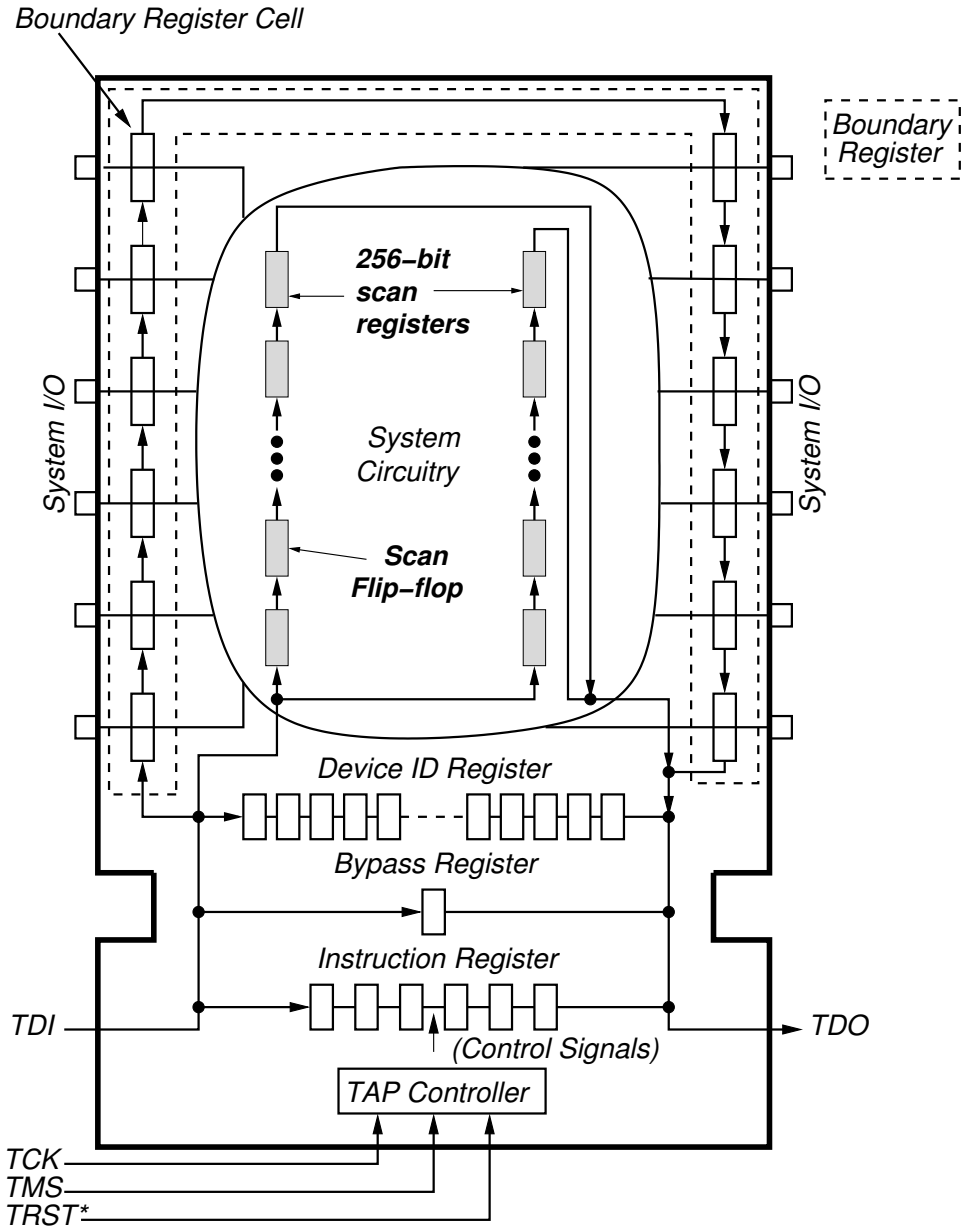
It takes 5 clocks to apply a pattern after shifting it in. March C− has 10 March elements.

$$\begin{aligned} \text{Test time (without shifting)} & \\ = \#\text{test patterns} &= 10 \times 4 \times 10^{28} = \boxed{10737418240} \\ \text{Total test time} &= \frac{\#\text{patterns}(\text{shift time} + 6) + \text{shift time}}{\text{clock rate}} \\ \text{Shift time} &= 64 \\ \text{Total test time} &= \frac{10737418240 \times 70 + 64}{200 \text{ MHz}} = \boxed{3758.1 \text{ s}} \end{aligned}$$

We assume the presence of the hardware in Chapter 16 (page 572 of the book), where the 4-bit word is made into a shift register to test multi-bit words.

16.10 Full scan and boundary scan

The following schematic shows the full-scan connections to the JTAG circuitry.



Chapter 17: Analog Test Bus Standard

17.1 Inductance measurement

$$\phi = LI, \qquad V = L\frac{dI}{dt}$$

One test method would be to apply a current ramp to the inductor while measuring the voltage across it, which should be constant. We will ignore the startup and shutdown transients and just look at the steady-state response.

Since all of the switches induce a voltage drop (due to their 100Ω resistance), we can first disable the inductor by applying a known DC current and measuring the voltage drop in the system. Then, we can calculate the efective series resitance:

$$V = IR_{eff}, \quad so,\ R_{eff} = \frac{I}{V}$$

For the current ramp test,

$$V = R_{eff}I(t) + L\frac{dI}{dt}$$

Let $I(t) = 0 + 50\ mA/s \times t,\ so\ \frac{dI}{dt} = 50\ mA/s$. Then,

$$V(t) = R_{eff} \times \frac{50\ mA}{s}t + L \times \frac{50\ mA}{s}$$

So,

$$L = \frac{V(t) - R_{eff}50\frac{mA}{s}t}{50\ mA/s}$$

or

$$L = \frac{V(t)}{50\ mA/s} - R_{eff}t$$

Test method (see Figure 17.8, page 586 of the book):

I. Measure R_{eff}

 A. Turn on $S5$, $S6$, $SB1(pin1)$, $SB2(pin1)$, $SG(pin1)$, and force $100\mu A$ of current into $AT1$ with ATE current source.

 B. Measure voltage V_1 between $AT2$ and *ground*.

 C. Turn off $SB2$ at $pin1$ and turn on $SB2$ at $pin2$.

 D. Measure voltage V_2 between $AT2$ and *ground*.

E. $V = V_1 - V_2$. If this is not 0, then there is significant current leakage through $AT2$ or there is series resistance in the inductor.

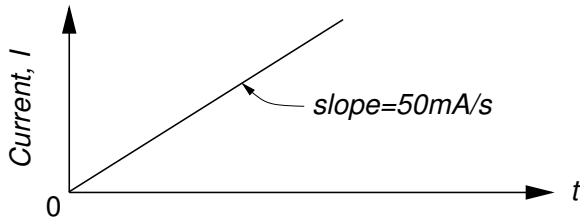
$$R_{eff} = \frac{V_1 - V_2}{100\mu A}$$

We assume the latter.

II. Measure L

A. Turn off $SB2$ at $pin2$ and turn on $SB2$ at $pin1$.

B. Apply this current ramp through $AT1$:



C. Measure V at two different time instants t_1 and t_2 (chosen so that the startup transient at $t = 0$ has died down.) Use steps I, B through D, to do this.

D. Compute L twice:

$$L = \frac{V(t) - R_{eff}\frac{50mA}{s}t}{50mA/s}$$

The measurements L_1 and L_2 should agree within $60 \times 10^{-10} H$. If they do not agree within that tolerance, reject the inductor. Otherwise, average the measurements:

$$L_{ave} = \frac{L_1 + L_2}{2}$$

Verify that $59.994\mu H \leq L_{ave} \leq 60.006\mu H$. If L_{ave} is not in this range, reject the inductor.

An error $V_{err}(t)$ in the $V(t)$ measurement changes the calculated L by

$$\Delta L = \pm\frac{V_{err}(t)}{50mA/s}$$

$$V_{err}(t) = 20\mu V, \quad so\ \Delta L = \frac{20\mu V}{50mA/s} = 400\mu H$$

Measurement error exceeds inductor tolerance. This is an unreliable test! We need to change the current slope.

To have $|\Delta L| \leq \frac{1}{2}60\mu H \times 10^{-4}$, we need

$$\frac{dI}{dt} = \frac{20\mu V}{|\Delta L|} = \frac{20\mu V}{\frac{1}{2}60\mu H \times 10^{-4}} = 6666.67\ A/s$$

Redesign the test with this $\frac{dI}{dt}$, and with sampling times t_1 and t_2 so that we only get $100mA$ maximum current.

$$t_1,\ t_2 < \frac{100mA}{6666.67\ A/s} = 0.150\mu s$$

This may or may not cause a problem with the startup transient being sampled.

Alternative solution:

Apply a sinusoidal current waveform to the inductor and measure the voltage across it, which should have some phase shift. Ignore startup and shutdown transients and just look at the steady-state response.

We assume that the system voltmeter has extremely high impedance, so that we can ignore the current through it and, therefore, we do not model $AB2\ R's$. Then:

$$\begin{aligned} V_1 &= I(100\Omega + j\omega L) \\ V_2 &= I(100\Omega) \\ V_1 - V_2 &= j\omega LI \\ Z &= j\omega L = \sqrt{\omega^2 L^2} = \omega L \\ L &= \frac{V_1 - V_2}{\omega I} \end{aligned}$$

Test method (see Figure 17.8, page 586 of the book):

I. Repeat prior part I to calculate R_{eff} (series resistance of inductor.)

II. Measure L

A. Turn off $SB2$ at $pin2$ and turn on $SB2$ at $pin1$.

B. Apply $I = \sin(2\pi \times 60t)$ through $AT1$.

C. Measure V_1 and V_2.

D. $V_1 - V_2 = I(R_{eff} + j\omega L)$, $\omega = 2\pi 60\ rad/s$

$$Z = R_{eff} + j\omega L = \sqrt{R_{eff}^2 + \omega^2 L^2}$$

So,

$$\frac{(V_1 - V_2)^2}{I^2} = R_{eff}^2 + \omega^2 L^2$$

$$L = \frac{1}{\omega}\sqrt{\frac{(V_1 - V_2)^2}{I^2} - R_{eff}^2}$$

Both R_{eff} and $V_1 - V_2$ have an error term proportional to voltmeter error. So, the type of analysis at the end of the first solution still applies.

17.2 Capacitance measurement

$$Q = CV, \quad I = C\frac{dV}{dt}$$

One test method would be to apply a voltage ramp to the capacitor while measuring the current across it, which should be constant. We will ignore the startup and shutdown transients and just look at the steady-state response.

Since all of the switches induce voltage drop (due to their 100Ω impedance), we can first disable the capacitor by applying a known DC voltage and measuring the current in the system. Then, we can calculate the effective series resistance.

$$I = \frac{V}{R_{eff}} \quad \text{so} \quad R_{eff} = \frac{I}{V}$$

For the voltage ramp test,

$$\begin{aligned} V &= IR_{eff} + \int_0^t \frac{Idt}{C} \quad \text{or} \\ \frac{dV}{dt} &= \frac{I}{C} \quad \text{(since } I \text{ will be constant)} \end{aligned}$$

So,

$$C = \frac{I}{dV/dt}, \text{ choose } \frac{dV}{dt} = 50mV/s$$

Test method (see Figure 17.8 on page 586 of the book):

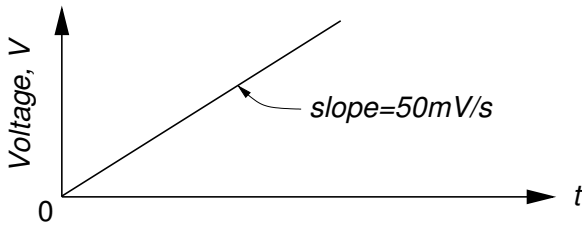
I. Optionally measure R_{eff} (not needed for accurate C measurement.) Method is the same as for Part I of solution to Problem 17.1.

II. Measure C

A. Turn on $S5$, $S6$, $SB1$ (pin 1), SG (pin 1), and $SB2$ ant pin 1.

B. Turn off $SB2$ at pin 2.

C. Apply the voltage ramp, shown below, at $AT1$.



D. Measure I at two different time instants t_1 and t_2 (chosen so that startup transient at $t = 0$ has died down.)

E. Compute C twice:

$$C = \frac{I}{dV/dt}$$

The measurements should agree within $0.02nF$. If they do not, reject the capacitor. Otherwise, average the measurements.

$$\begin{aligned} C_{ave} &= C_1 + C_2 \\ 199.98nF &\leq C_{ave} \leq 200.02nF \end{aligned}$$

If not, reject the capacitor.

Alternative solution:

Apply a sinusoidal current waveform to the capacitor and measure the voltage in the circuit, which should have some phase shift. Ignore startup and shutdown transients, and just look at the steady state response. Ignore impedances of $S6$ and $SB2$ in $AT2$, since the voltmeter has very high impedance.

$$\begin{aligned} V_1 &= I(100\Omega + \frac{1}{j\omega C}) \\ V_2 &= I(100\Omega) \\ V_1 - V_2 &= \frac{I}{j\omega C} \\ Z &= \frac{1}{j\omega C} = \frac{j\omega C}{-\omega^2 C^2} = \sqrt{\frac{\omega^2 C^2}{\omega^4 C^4}} = \frac{1}{\omega C} \\ C &= \frac{I}{\omega C(V_1 - V_2)} \end{aligned}$$

Test method to measure C (see Figure 17.8 on page 586 of the book):

A. Turn off $SB2$ at pin 2 and turn on $SB2$ at pin 1.

B. Turn on $S5$, $SB1$ at pin 1, and SG at pin 2.

C. Apply $I = \sin(2\pi \times 60t)$ through $AT1$.

D. Measure V_1 and V_2,

$$\begin{aligned} Z &= \frac{1}{\omega C}, \quad so \\ \frac{(V_1 - V_2)^2}{I_2} &= \omega^2 C^2 \\ C &= \sqrt{\frac{(V_1 - V_2)^2}{\omega I}} \end{aligned}$$

17.3 Resistance measurement

To measure a resistance R of value $40\Omega \pm 10\%$, we proceed as follows (see Figure 17.8 on page 586 of the book):

A. Turn on $S5$, $S6$, $SB1$ (pin 1), $SB2$ (pin 2), SG (pin 1) and force $100\mu A$ of current into $AT1$ with ATE current source ($AT1$ connects to $AB1$, $AT2$ connects to $AB2$.)

B. Measure voltage V_1 between $AT2$ and *Ground*.

C. Turn off $SB2$ at pin 1 and turn on $SB2$ at pin 2.

D. Measure voltage V_2 between $AT2$ and *Ground*.

E. $V = V_1 - V_2$ and $R_{eff} = V_1 - V_2$. This will be correct only if the ATE system voltmeter has extremely high impedance (so that current flow in $AT2$ can be ignored.) If that holds, there is no voltage drop in $SB2$ at pins 1 and 2 and at $S6$ in $AT2$. However, notice that the external R that we are measuring (40Ω) is smaller than 100Ω, the switch resistance.

First, we calculate the effect of voltmeter measurement error of $20\mu V$:

$$\begin{aligned} V_{1eff} &= V_1 + 20\mu V \\ V_{2eff} &= V_2 - 20\mu V \\ R_{eff} &= \frac{V_1 \pm 20\mu V - (V_2 \mp 20\mu V)}{100\mu A} \\ &= \frac{V_1 - V_2}{100\mu A} \pm 0.4\Omega \end{aligned}$$

The error of 0.4Ω equals the tolerance on the resistor value of 40Ω. Although this test can be used, it will most likely reject the vast bulk of resistors tested. A better test would be to reduce the ATE system voltmeter measurement error to $2\mu V$. Alternatively, the circuit under test can be redesigned not to use such low-valued R's.

We should now calculate a bound on the leakage current I_{leak} in the voltmeter on $AT2$.

$$\begin{aligned} V_{1AT2} &= V_1 - (R_{SB2_{p1}} + R_{S6})I_{leak} \\ V_{2AT2} &= V_2 - (R_{SB2_{p2}} + R_{S6})I_{leak} \\ V_{1AT2} - V_{2AT2} &= V_1 - V_2 - I_{leak}(R_{SB2_{p1}} + R_{S6} - R_{SB2_{p2}} - R_{S6}) \\ V_1 - V_2 &= V_{1AT2} - V_{2AT2} - I_{leak}(R_{SB2_{p1}} - R_{SB2_{p2}}) \\ \text{Error} &= \pm I_{leak}(R_{SB2} \text{ mismatch between } \textit{pins } 1 \text{ and } 2) \end{aligned}$$

We should keep this error to be $\frac{1}{10}$ of the system voltmeter error, or $2\mu V$. The following table shows permitted combinations:

R_{SB2} mismatch	Upper bound on I_{leak}
10Ω	$0.2\mu A$
5Ω	$0.4\mu A$
2Ω	$1.0\mu A$
1Ω	$2.0\mu A$
0.1Ω	$20.0\mu A$

17.4 Analog core test

The difference between *INTEST* and *PROBE* instructions for testing an analog core is as follows. *PROBE* does not require boundary scan hardware on lines from digital circuits to analog circuits, whereas *INTEST* does. As a result, the *PROBE* instruction lets internal digital circuits interact with the analog core, while *INTEST* disconnects the digital core and replaces it with set up patterns from the boundary register.

17.5 Bus impedance calibration

Measurement of resistive impedance in $AT1$ and $AT2$ buses:

1. In the *test bus interface circuit* (TBIC), close switch $S1$ connecting $AT1$ to V_H. Ground the other end of $AT1$ at the automatic test equipment (ATE). Measure the current in $AT1$, $I = (V_H - 0)/R_{AT1}$. So, $R_{AT1} = V_H/I$.
2. Repeat step 1, but using $AT2$ and closing switch $S2$. Thus, $R_{AT2} = V_H/I$.
3. In the TBIC, close switch $S3$ to connect $AT1$ to V_L. Set the other end of $AT1$ to V_{DD} at the ATE. Measure the current in $AT1$, $I = (V_{DD} - V_L)/R_{AT1}$. So, $R_{AT1} = (V_{DD} - V_L)/I$. Compare this value to the result of step 1.
4. Repeat step 3, but using $AT2$ and closing switch $S4$ in the TBIC. Thus, $R_{AT2} = (V_{DD} - V_L)/I$. Compare this value to the result of step 2.

An alternative procedure:

1. Disconnect all pins from $AB1$.
2. In the TBIC. close switches $S5$ and $S8$ to connect both $AT1$ and $AT2$ together through $AB1$.
3. Drive $AT1$ with a voltage V_1 and drive $AT2$ with V_2. Measure the current I flowing into $AT2$ at the ATE:

$$I = \frac{V_1 - V_2}{R_{total}}$$

So,

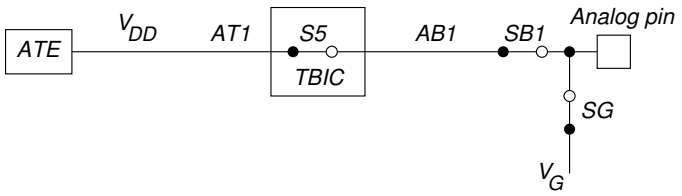
$$R_{total} = \frac{V_1 - V_2}{I}$$

4. Repeat steps 1 through 3, but using $AB2$ and closing switches $S6$ and $S7$.

17.6 Bus impedance calibration

1. First, measure resistance in $AT1$ and $AT2$ switches $S5$ through $S8$.

2. At a given pin, connect $AB1$ by closing $SB1$ and close SG for that pin. All other pin switches should be open. Connect $AT1$ to $AB1$. Force V_{DD} into $AT1$ at the automatic test equipment (ATE). Measure I at $AT1$:



$$I = \frac{V_{DD} - V_G}{R_{S5} + R_{SB1} + R_{SG}}$$

So,

$$R_{SB1} + R_{SG} = \frac{V_{DD} - V_G}{I} - R_{S5}$$

Assume:

$$R_{S5} = \frac{R_{S5} + R_{S8}}{2}, \text{ from Problem 17.5}$$

3. Repeat step 2 closing SL at the pin rather than SG. Then,

$$R_{SB1} + R_{SL} = \frac{V_{DD} - V_L}{I} - R_{S5}$$

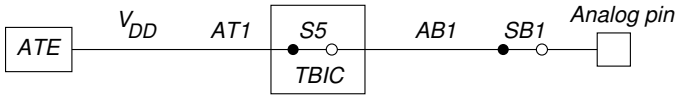
4. Repeat steps 1 through 3, but using $AB2$ instead of $AB1$ and R_{S7} rather than R_{S5}.

Alternative method:

1. Probe the analog pin with the ATE.

2. Measure resistance in $AT1$ and $AT2$ switches $S5$ through $S8$.

3. At the pin, connect $AB1$ by closing $SB1$ for that pin. All other pin switches should be open. Connect $AT1$ to $AB1$. Force V_{DD} into $AT1$ at the ATE. Make the ATE force the analog pin to ground. Measure I at $AT1$.



$$
\begin{aligned}
I &= \frac{V_{DD}}{R_{S5} + R_{SB1}} \\
R_{SB1} &= \frac{V_{DD}}{I} - R_{S5} \\
R_{S5} &= \frac{R_{S5} + R_{S8}}{2}, \text{ from Problem 17.5}
\end{aligned}
$$

4. Repeat step 3, but using $AB2$ instead of $AB1$ and R_{S7} rather than R_{S5}.

Chapter 18: System Test and Core-Based Design

18.1 Fault dictionary

With faults a_1 and c_0 simultaneously present, the output function of the circuit of Figure 18.2(a) becomes $e = b$. Of the four tests shown on the Karnaugh map of Figure 18.2(b), only T_1 will show a failure at the output. Thus, the test syndrome is 1000. This matches with the third row of the fault dictionary given in Table 18.1. **We diagnose the fault as $\mathbf{a_1}$.**

Note: We fail to detect the other fault c_0 because its test $T_2 = (011)$ is invalidated by a_1.

18.2 Diagnosis

The fault (a_1, b_1) makes the faulty function of the circuit of Figure 18.2 a constant 1, i.e., $e = 1$. Using the diagnostic tree of Figure 18.3, we first apply the test T_4. According to the Karnaugh map of Figure 18.2, the expected output is 1, which is the same as the faulty output. Thus, T_4 passes, and $t_4 = 0$. So, we follow the upper branch in the diagnostic tree and apply the test T_1 next. The expected output is 0 now and the test shows a failure. We take the lower branch and apply T_3, which also shows a failure, i.e., $(t_3 = 1)$. Following the lower branch, the procedure terminates giving **a diagnosis of three suspected single faults, $\mathbf{c_1}, \mathbf{d_1}$ and $\mathbf{e_1}$.**

Note: The suspected fault set does not contain the actual multiple fault (a_1, b_1). This is a failure of the diagnostic procedure based on single faults. One can verify that the use of the fault dictionary will also lead to the same erroneous diagnosis.

18.3 Diagnosis

Detection of all stuck-at faults in the exclusive-OR circuit of Figure 18.8 requires all four vectors. The following table shows inputs and outputs for the circuit.

Test	Input vector ab	Output i												
		no-fault	a_0	a_1	b_0	b_1	c_0	c_1	e_1	f_0	f_1	d_0	i_0	i_1
T_1	00	0	0	**1**	0	**1**	0	0	0	0	**1**	0	0	**1**
T_2	01	1	1	**0**	**0**	1	1	**0**	1	**0**	1	1	**0**	1
T_3	10	1	**0**	1	1	**0**	1	1	**0**	1	1	**0**	**0**	1
T_4	11	0	**1**	0	**1**	0	**1**	0	0	0	0	0	0	**1**

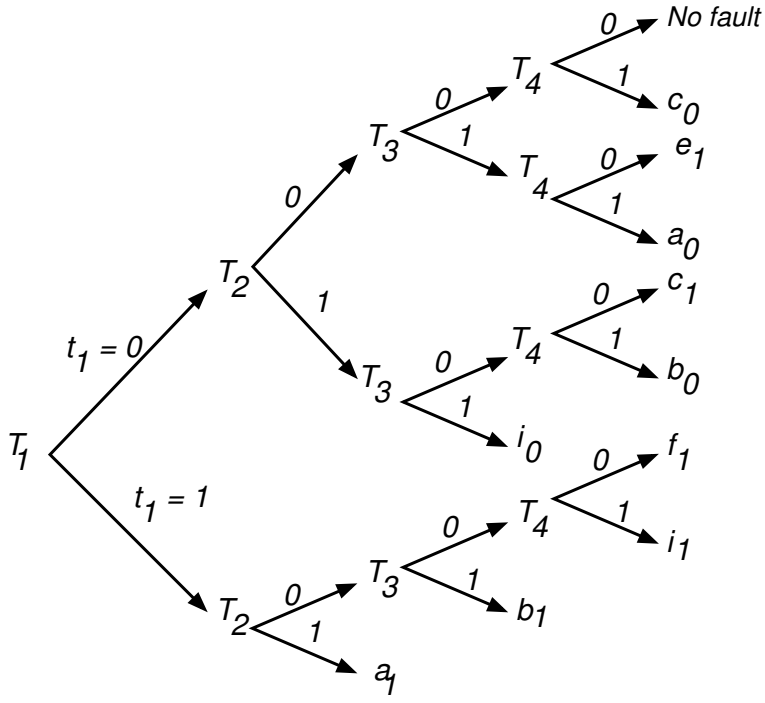
Here a_j denotes the fault a s-a-j, and **boldface** entries mark the outputs that defer from the correct outputs and hence indicate fault detection. We have collapsed some faults. For example, c_0 represents the equivalent fault set $[c_0, e_0, g_0]$. Similarly, f_1 represents the equivalent set $[f_1, d_1, h_0]$, and i_0 represents the equivalent set $[g_1, h_1, i_0]$. In the above table, we further notice that faulty functions are identical for c_1 and f_0, and for e_1 and d_0. Therefore, faults f_0 and d_0 can be dropped due to equivalence. The next table gives the fault dictionary with faults arranged in the order of increasing numerical value of syndrome.

Fault	Test syndrome			
	t_1	t_2	t_3	t_4
No fault	0	0	0	0
c_0	0	0	0	1
e_1	0	0	1	0
a_0	0	0	1	1
c_1	0	1	0	0
b_0	0	1	0	1
i_0	0	1	1	0
f_1	1	0	0	0
i_1	1	0	0	1
b_1	1	0	1	0
a_1	1	1	0	0

Where $t_k = 1$, if test T_k detects the fault, else it is 0.

18.4 Diagnosis

A diagnosis tree constructed from the above fault dictionary is shown in the following figure.



18.5 Diagnosis

We find the average number of tests for diagnosis using Equation 18.2. Since the probability of occurrence of faults a_1, b_1, c_0 and e_0 is 0.25, from the diagnostic tree of Figure 18.3, we get

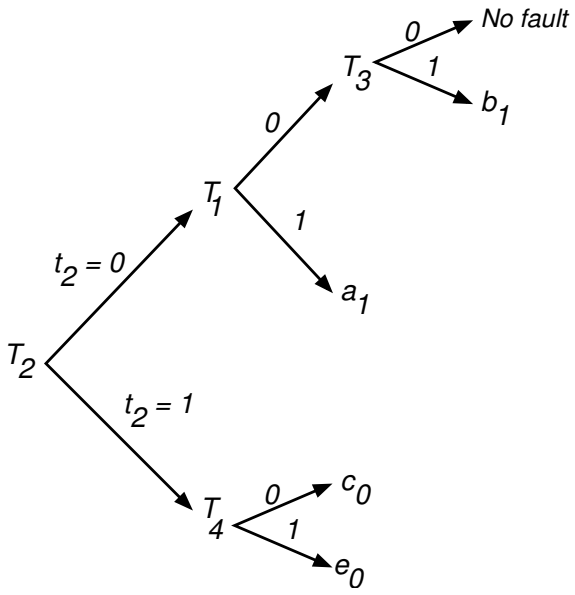
Average diagnostic test length

$$
\begin{aligned}
&= 3 \times Prob(a_1) + 4 \times Prob(b_1) + 3 \times Prob(c_0) + 2 \times Prob(e_0) \\
&= 3 \times 0.25 + 4 \times 0.25 + 3 \times 0.25 + 2 \times 0.25 \\
&= 3
\end{aligned}
$$

The diagnosis of a faulty circuit will terminate on average after 3 tests.

18.6 Diagnosis

We will only consider the four faults that have non-zero probabilities of occurrence. A binary search generally leads to an optimum diagnosis since each test provides a pass/fail result. Examining the tests of Table 18.1, we find that application of T_2 divides the fault set into subsets (c_0, e_0) and (a_1, b_1). Similar consideration leads to the following diagnostic tree.



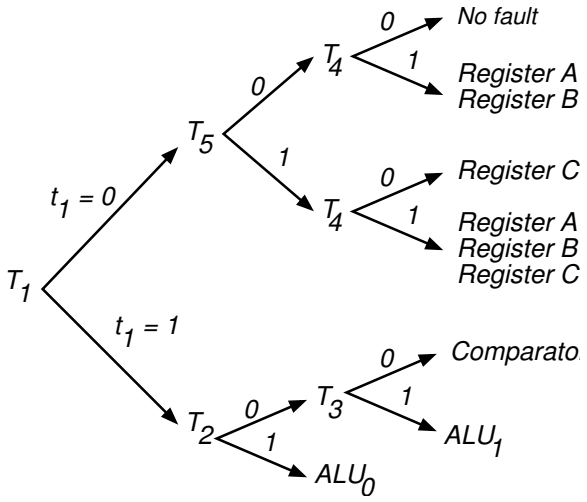
Average diagnostic test length

$$
\begin{aligned}
&= 2 \times Prob(e_0) + 2 \times Prob(c_0) + 2 \times Prob(a_1) + 3 \times Prob(b_1) \\
&= 3 \times 0.25 + 4 \times 0.25 + 3 \times 0.25 + 2 \times 0.25 \\
&= 2.25
\end{aligned}
$$

The average diagnosis time is 2.25 times that of applying one test.

18.7 Diagnosis

A diagnosis tree for the system of Figure 18.4 and its tests shown in Table 18.3 is shown below.



We first apply test T_1. If it fails, i.e., $t_1 = 1$, we presume that either the ALU's or the comparator are faulty. So, we apply T_2. A failure now means $ALU0$ is faulty. Otherwise, we apply T_3, which either fails if $ALU1$ is faulty, or passes if the comparator is faulty.

If T_1 passes, we assume that ALU's are not faulty and the $x's$ is the syndromes of ALU's under t_4 and t_5 can be changed to 0. Also, the rest of the procedure assumes that only one unit is faulty. We apply T_5 followed by T_4. If T_5 fails and T_4 passes then *Register C* is considered faulty. If T_5 passes and T_4 fails then both *Register A* and *Register B* become suspects. If both tests fail then we conclude that perhaps *Register C* and at least one among the other two registers are faulty.

The diagnostic tree shows that the procedure terminates with the application of three or fewer tests.

18.8

(a) Since the processor is used to test other modules, its correctness should be ascertained first. A self-test is, therefore, desirable. If the self-test of the processor reports a failure then the rest of the tests need not be conducted.

(b) If the random-vector coverages in the ASIC, FSM and DSP are low, additional vectors can be algorithmically generated to cover the undetected faults in those modules. These vectors can be loaded in the RAM (after its own self-test has been successfully completed) and applied to the appropriate module by the processor. Their responses can also be compacted in the same way as the processor compacts the responses of the random vectors.

Alternatively, the additional vectors can be applied by an external tester via the on-chip bus.

(c) Since there is a DSP module on the chip it can be used to test the analog module.

18.9 Rent's rule

Suppose that T_i is the number of I/O terminals and G_i is the number of gates for ith circuit. The estimated number of terminals for a circuit with G_i gates is,

$$T(G_i) = KG_i^{\alpha}$$

We compute the squared-error as,

$$SE \;=\; \sum_{i=1}^{10} \left(\frac{T(G_i) - T_i}{T_i} \right)^2 = \sum_{i=1}^{10} \left(\frac{KG_i^{\alpha}}{T_i} - 1 \right)^2$$

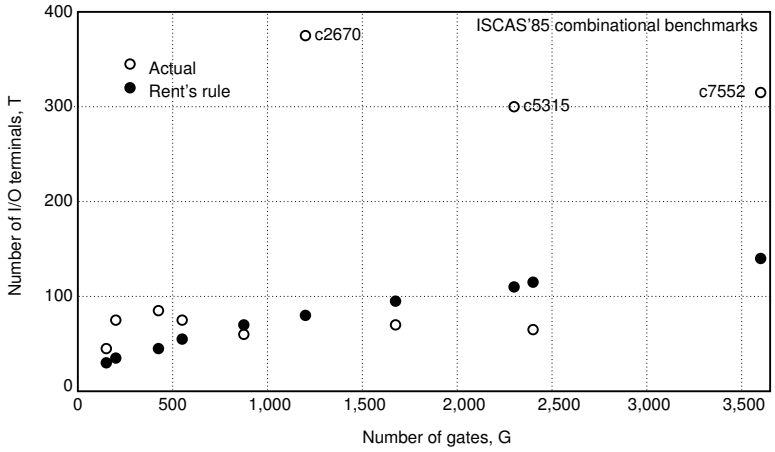
where the sum is carried over 10 benchmark circuits. To minimize the square-error, we proceed as follows:

$$\frac{dSE}{dK} = 2 \sum_{i=1}^{10} \left(\frac{KG_i^{\alpha}}{T_i} - 1 \right) \frac{KG_i^{\alpha}}{T_i} = 0$$

Simplification leads to,

$$K = \frac{\sum_{i=1}^{10} \frac{G_i^{\alpha}}{T_i}}{\sum_{i=1}^{10} \left(\frac{G_i^{\alpha}}{T_i} \right)^2}$$

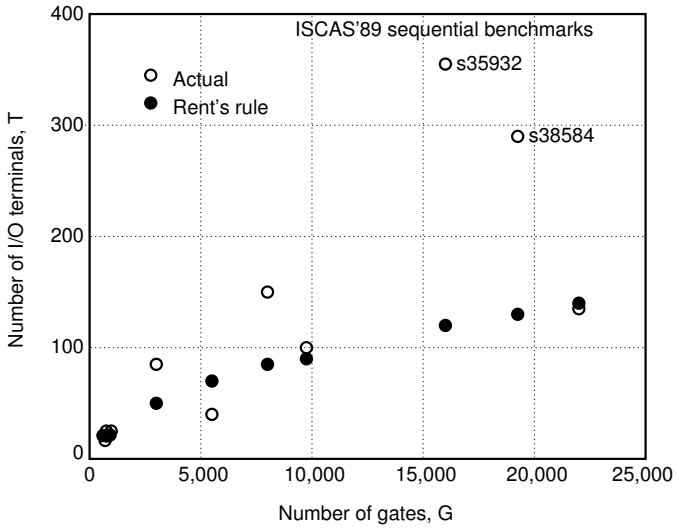
Using $\alpha = 0.5$ and the values for T_i (sum of PI and PO) and G_i (gates) from Table 18.5, we evaluate $K = 2.32$. The given data and the computed values of $T = 2.32\sqrt{G}$ are plotted on the following graph.



We notice that three circuits, c2670, c5315 and c7552, have much larger number of terminals than that estimated by the Rent's rule formula. This is because perhaps these are combinational parts of scan circuits. Our value of K is strongly influenced by the majority of circuits that are non-scan type and have fewer terminals. To represent scan circuits, K will have to be higher.

18.10 Rent's rule

We substitute the data from Table 18.6 in the formula for K derived in the previous problem. The calculation gives $K = 0.93$. This smaller value of K (as compared to 2.32 obtained for combinational circuits) is somewhat typical of sequential circuits. Actual data and computed values from $T = 0.93\sqrt{G}$ are shown on the following graph.



Note that Rent's estimates are highly approximate and depend on the style of design. We find large error in the cases of s35932 and s38584.

Final Exam: Graduate Course – Testing of Ultra Large Scale Integrated Circuits

Rutgers Univ., 16:332:576, Spring 2002

May 15, 2002

Problem 1: Fault Modeling and Simulation (11 Points)

For the circuit of Figure 1, do the following:

(i) Count the total number of single fault sites. (2 points)

(ii) Show that the three single faults, H s-a-1, J s-a-1, and K s-a-1, are equivalent. (4 points)

(iii) Using the parallel fault simulation algorithm (assuming a four-bit machine word), or any alternative fault simulation algorithm, determine which of the three single faults, F s-a-1, H s-a-0, and L s-a-1, are detected by the input vector $A = 1, B = 0$. (5 points)

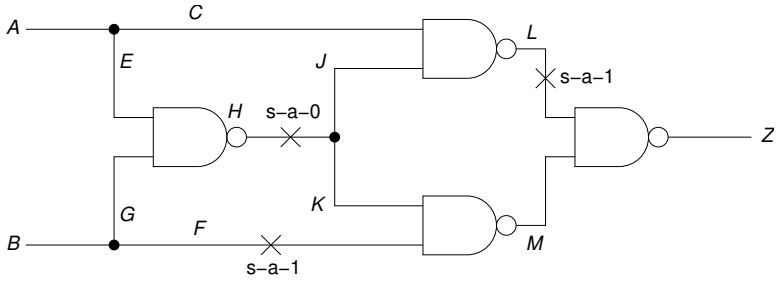


Figure 1: Circuit for fault modeling and simulation problem.

Solution to Problem 1

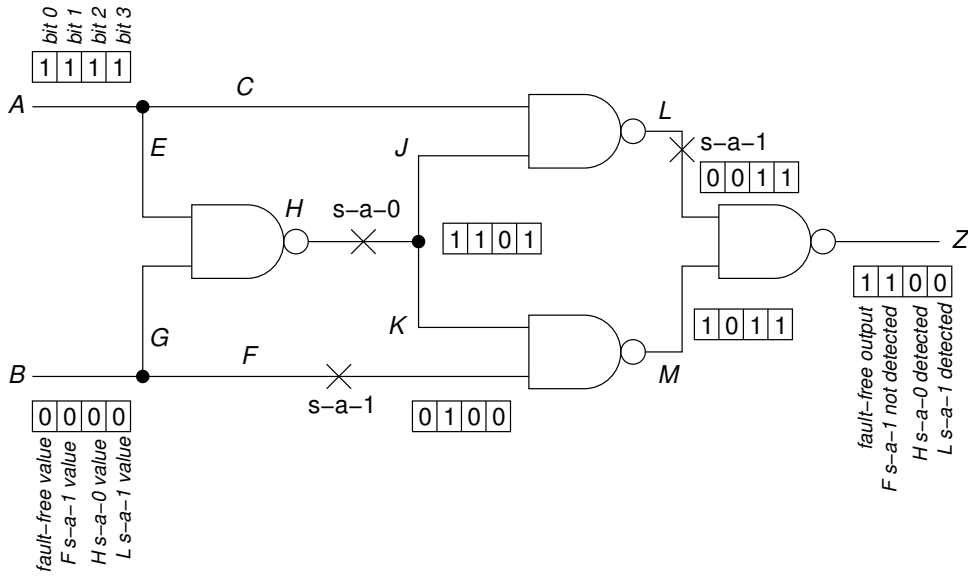
(1) Total fault sites $= \#PI + \#gates + \#fanout_branches = 2 + 4 + 6 = 12$

(ii) Three faulty functions are:

$$\begin{aligned} Z(H\ s-a-1) &= \overline{\overline{A}.\overline{B}} = A + B \\ Z(J\ s-a-1) &= \overline{\overline{A}.\overline{AB}.B} = \overline{\overline{A}.\overline{(\overline{A}+\overline{B})B}} \\ &= \overline{\overline{A}(A+\overline{B})} = \overline{\overline{A}\ \overline{B}} = A + B \\ Z(K\ s-a-1) &= \overline{\overline{A.\overline{AB}}.\overline{B}} = \overline{\overline{A(\overline{A}+\overline{B})}.\overline{B}} \\ &= \overline{(\overline{A}+\overline{B})\overline{B}} = \overline{\overline{A}\ \overline{B}} = A + B \end{aligned}$$

Since the faulty functions are identical, the three faults are equivalent. ■

(iii) Parallel fault simulation using a four-bit computer word is shown in the figure below. The fault-free value of a line is represented by bit 0, the value corresponding to fault F s-a-1 by bit 1, the value corresponding to fault H s-a-0 by bit 2, and the value corresponding to fault L s-a-1 by bit 3.



Simulated value of the output Z indicates that faults H s-a-0 and L s-a-1 are detected and F s-a-1 is not detected.

Problem 2: Combinational ATPG (11 Points)

Consider the circuit under test (CUT) with two outputs, C and S, shown in Figure 2. We desire a test that will detect a target fault at any one output but not at both outputs. Figure 2 gives an ATPG model circuit in which the two outputs of the CUT are combined into an exclusive-OR gate (shown shaded) to produce an output Z.

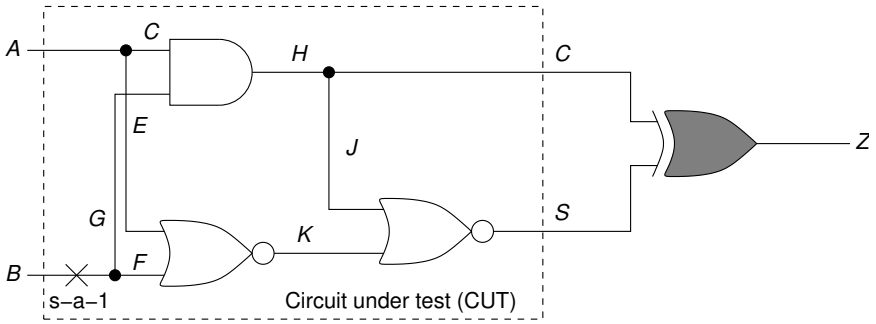


Figure 2: Circuit for combinational ATPG problem.

(i) Show that when a fault is detected at Z in the ATPG model, it must be exclusively detected either at C or at S. (4 points)

(ii) Using either the five-valued logic or the nine-valued logic, obtain a test to detect B s-a-1 exclusively at one of the outputs of the CUT. (4 points)

(iii) Can such a test be found for H s-a-0. (3 points)

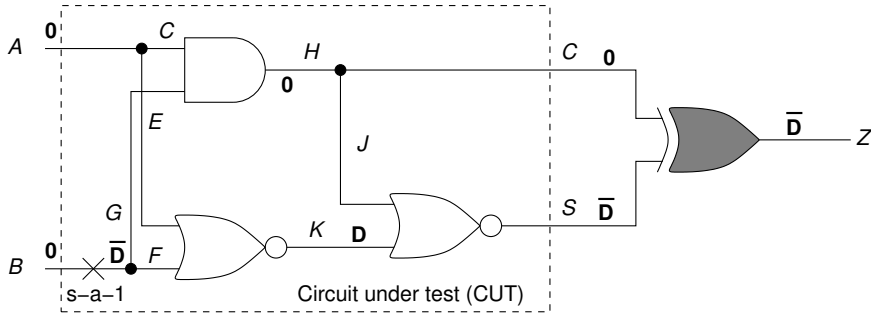
Solution to Problem 2

(i) The following table shows how the fault effect (D or $\overline{D}$) propagates through an exclusive-OR gate.

One input	Other input	Output
D	D	0
$\overline{D}$	$\overline{D}$	0
D	$\overline{D}$	1
D	0	D
$\overline{D}$	0	$\overline{D}$
D	1	$\overline{D}$
$\overline{D}$	1	D

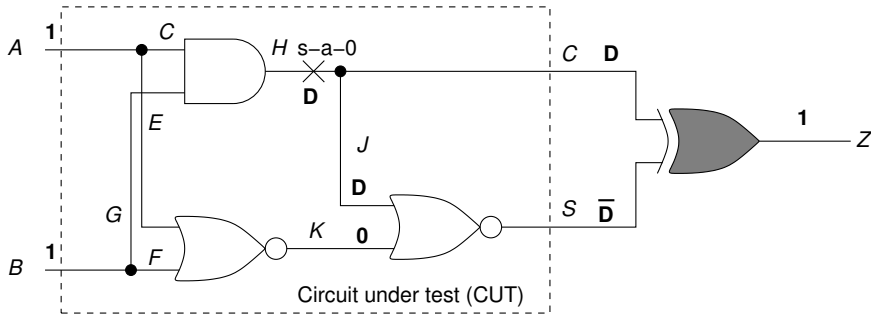
That is, for a fault to affect the value of Z, it should affect the value of C, or that of S, but not those of both. Therefore, when a test detects a fault at Z the fault must be detected (observable) exclusively at C or S.

(ii) Using the five-valued logic a test $A = 0,\ B = 0$ is found to detect the fault B s-a-1 at Z, as shown in the following figure. This test detects the fault at S, but does not detect it at C.



Test generation for B s–a–1 using five–valued logic.

(iii) The following figure shows that the fault H s-a-0 cannot be detected at Z. Only on input vector, $A = B = 1$, can activate the fault but it propagates the fault to both outputs of the CUT. An exclusive test for this fault is not possible.



No test can detect H s–a–0 at Z.

Problem 3: Testability Measures (11 Points)

For the circuit of Figure 3, compute the combinational and sequential SCOAP testability measures (both controllability and observability, and including the $CLOCK$ and **synchronous** $\overline{RESET}$ signals.)

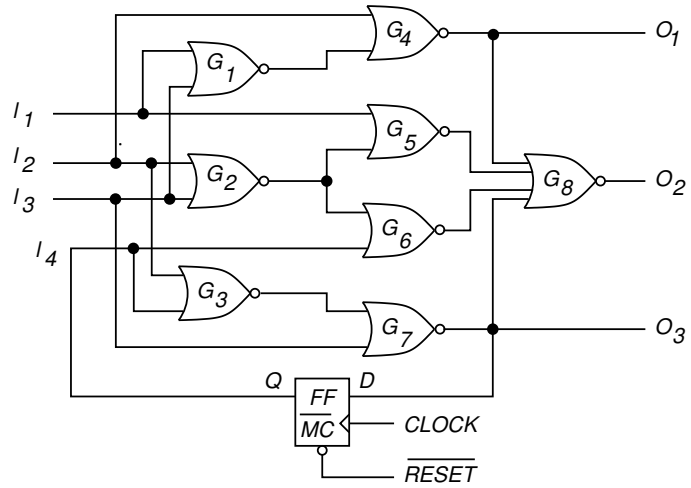
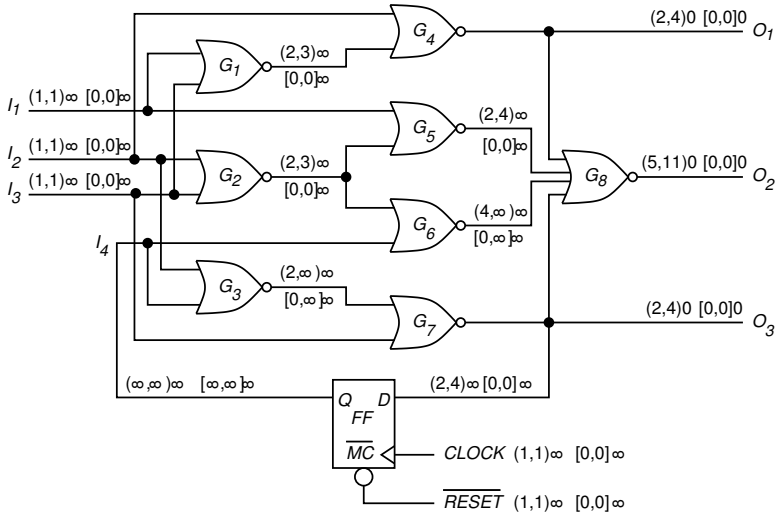


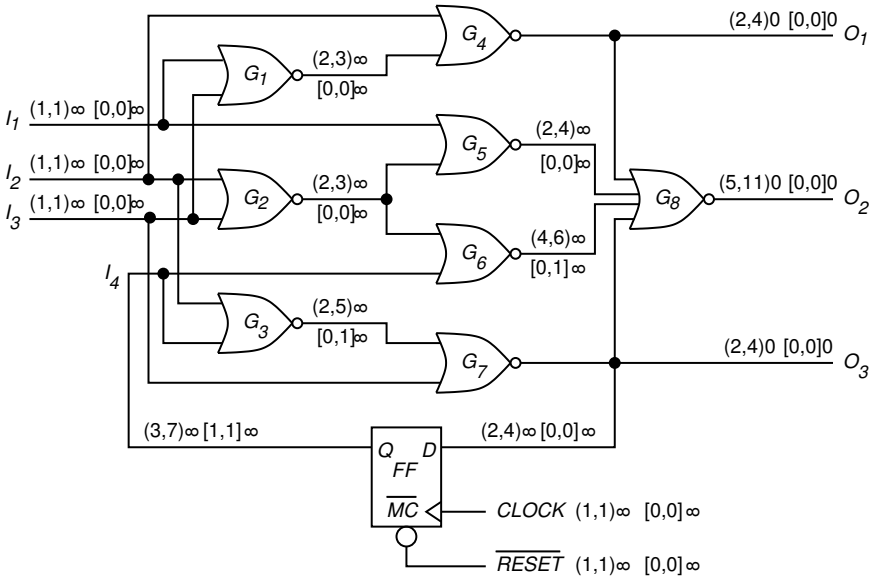
Figure 3: Circuit for Problem 3.

Solution to Problem 3

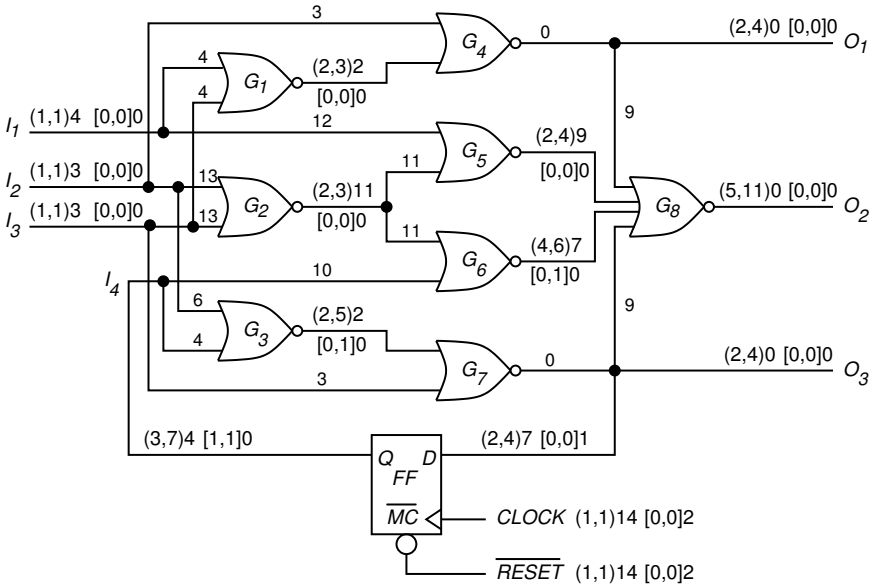
The steps of calculation for SCOAP testability measures are shown in the three figures that follow. Combinational measures are shown as $(CC0, CC1)CO$ and sequential measures as $[SC0, SC1]SO$.



Circuit of Problem 3: PI and PO initialization and first controllability pass.



Circuit of Problem 3: Converged controllability values.

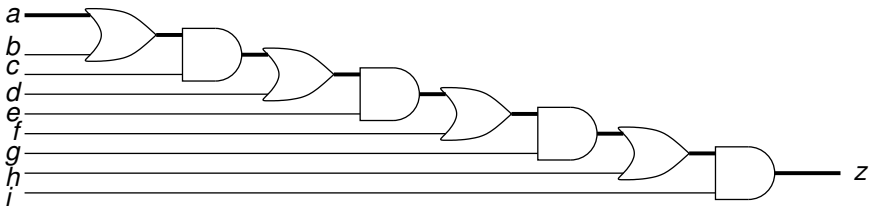


Circuit of Problem 3: All controllability and observability values.

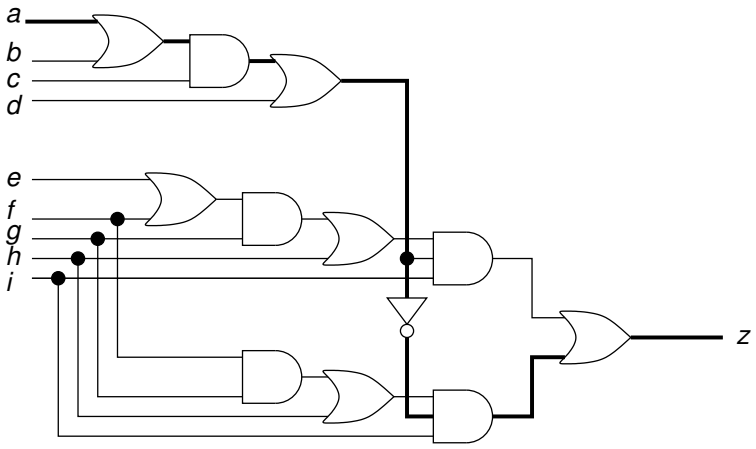
Problem 4: Delay Test

(i) Specify a single input change (SIC) test for the critical path $\uparrow a - z$ (shown in bold lines) in the circuit of Figure 4(i). Is this a robust test? (5 points)

(ii) The circuit of Figure 4(i) is redesigned in Figure 4(ii) to reduce the delay. Will the SIC test obtained above still test the longest delay path shown in bold lines? If not, what is the minimum modification required in the test? (6 points)



(i) Original circuit.



(ii) Redesigned circuit.

Figure 4: Circuits for delay test problem.

Solution to Problem 4

(i) A SIC test is found by statically sensitizing the path $a - z$ and applying a rising transition to a. Thus, the test is: $a = R1$, $b = S0$, $c = S1$, $d = S0$, $e = S1$, $f = S0$, $g = S1$, $h = S0$, $i = S1$. This is a robust test.

(ii) When the above test is applied to the redesigned circuit, the OR gate at the output receives a rising transition ($R1$) at its upper input and a $S0$ at the lower input. Although $R1$ is produced at Z, this transition does not arrive through the longest path (shown in bold). Thus, the upper path, which is 5-gate long,

is tested but the 6-gate path (shown as critical path) is not tested. We find that no input vector can sensitize this path when $a = 1$ and the PDF $\uparrow a - z$ for the critical path is untestable.

This part not required: When $a = 0$, setting $h = S1$ and leaving all other inputs as before, we can sensitize the critical path. Thus, the PDF $\downarrow a - z$ for the critical path can be tested by applying $a = F0$, $h = S1$ and leaving all other inputs as before. We also note that the s-a-1 fault at the output of the NOT gate is redundant. If that fault is removed, then the 6-gate critical path will also be removed. Then, the 5-gate path becomes the critical path and it is robustly testable by the test derived in part (i).

Problem 5: Memory Test (11 Points)

Rigorously prove that the MARCH C− test detects all inversion coupling faults $<\uparrow;\updownarrow>$. Indicate the testing time complexity for MARCH C− in terms of n, the number of bits in the memory.

MARCH C−	$\{ \Updownarrow (w0); \Uparrow (r0, w1); \Uparrow (r1, w0);$ $\Downarrow (r0, w1); \Downarrow (r1, w0); \Updownarrow (r0) \}$

Solution to Problem 5

We rigorously prove that the MARCH C− test detects all inversion coupling faults (CFin).

The MARCH C− test is,

$$\{ M0: \Updownarrow (w0); M1: \Uparrow (r0, w1); M2: \Uparrow (r1, w0);$$
$$M3: \Downarrow (r0, w1); M4: \Downarrow (r1, w0); M5: \Updownarrow (r0) \}$$

and the inversion coupling faults are $<\uparrow;\updownarrow>$ and $<\downarrow;\updownarrow>$.

Necessary condition: For all cells that are coupled, each should be read after series of possible CFins may have occurred, and the number of coupled cell transitions must be odd.

***Fault* $<\uparrow;\updownarrow>$:** *Address of coupled cell i > address of coupling cell j.* Cell j initialized to 0 by $M0$, j is made to $\uparrow$ by $M1$, coupled cell i set to 0 by $M0$, unexpected inversion detected by $M1$, number of coupled cell inversions = 1. *Address of coupled cell i < address of coupling cell j.* Cell j initialized to 0 by $M2$, j made to $\uparrow$ by $M3$, coupled cell i set to 0 by $M2$, unexpected inversion detected by $M3$, number of coupled cell inversions = 1.

That completes the proof. ■

The test complexity is $O(10n)$.

Problem 6: Analog Test (11 Points)

a. *Unit test period.* For an analog circuit, the test waveform frequency $F_t = 2010\ Hz$ and the sampling frequency in the DSP ATE is 8000 s/s. Compute the minimum *unit test period* and the corresponding *primitive frequency.*

b. *Unit test period.* A CODEC is to be tested on a DSP ATE with $F_s = 8000\ s/s$. Originally, $P = 40\ ms$, but that does not allow $N \geq 400$. Select a test waveform frequency as close to 2000 Hz as possible that still generates $N \geq 400$ unique samples by adjusting Δ. How many test waveform cycles (M) will there be in the *primitive period*?

Solution to Problem 6

a.

$$\frac{F_t}{F_s} = \frac{M}{N} = \frac{2,010Hz}{8,000Hz} = \frac{201}{800}$$

That is, $M = 201$ and $N = 800$. Unit test period is obtained as,

$$\begin{aligned} \text{UTP} &= \frac{M}{F_t} = \frac{201}{2,010Hz} = 0.1\ sec \\ \text{Primitive frequency, } \Delta &= \frac{1}{UTP} = 10Hz \end{aligned}$$

b.

$$\begin{aligned} \Delta &= \frac{1}{p} = \frac{1}{40\ msec} = 25Hz \\ N &= \frac{F_s}{\Delta} = \frac{8,000\ s/s}{25Hz} = 320 \end{aligned}$$

We must change Δ and p to get $N = 400$.

$$\Delta = \frac{8,000\ s/s}{400} = 20Hz$$

If $F_t = 2,000Hz$, $M = \frac{F_t}{\Delta} = \frac{2,000Hz}{20Hz} = 100$, and $\frac{M}{N} = \frac{100}{400} = \frac{1}{4}$, which gives only 4 samples. So, choose either $M = 99$ or $M = 101$.

For $M = 99$, $F_t = M \times \Delta = 99 \times 20Hz = 1,980Hz$, and

For $M = 101$, $F_t = M \times \Delta = 101 \times 20Hz = 2,020Hz$.

In both cases, we get 400 unique samples.

Problem 7: DFT (11 Points)

The circuit in Figure 5 is a sequence detector. A sequence 111 in the $INPUT$ bit-stream locks the output Z to 1. The state of the circuit can be set to 000 with output $Z = 0$ by applying $CLEAR = 1$.

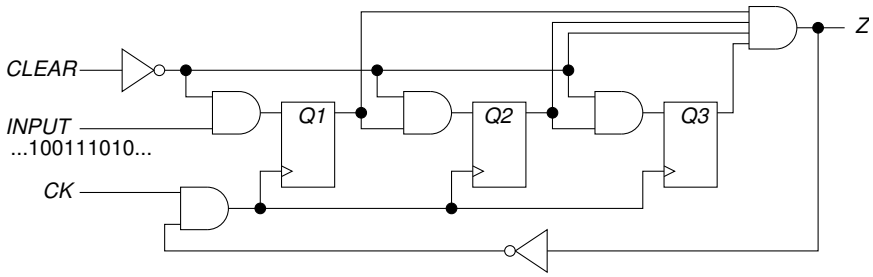


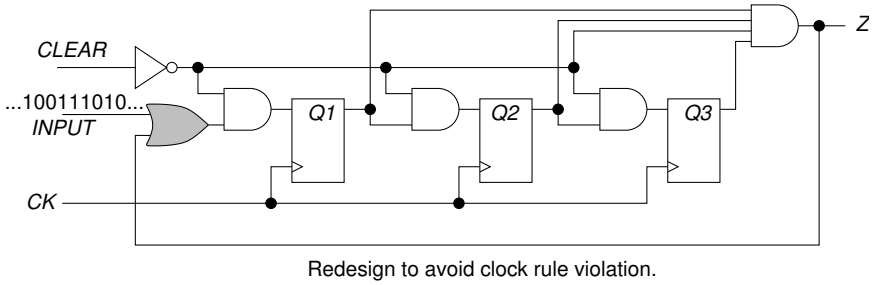
Figure 5: Circuit for DFT Problem 7.

(i) Redesign the circuit using minimum extra hardware to conform to the scan design rule, "clock must not be gated by a combinational signal." Neatly sketch the redesigned circuit. (5 points)

(ii) Sketch the schematic of the full-scan circuit using the multiplexer type of scan flip-flops (SFFs). Show the complete wiring of the $SCANIN$, $SCANOUT$ and test control (TC) signals. (6 points)

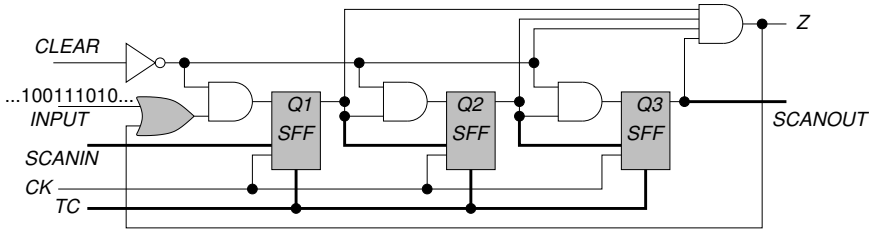
Solution to Problem 7

(i) To conform to the "clock rule," we use the clock gating signal Z to inhibit the data input. This is economically done by inserting an OR gate (shown shaded) in the following schematic. $Z = 0$ allows the normal $INPUT$ to be applied to the circuit. $Z = 1$, which occurs only when a 111 stream is detected, forces a constant 1 input.

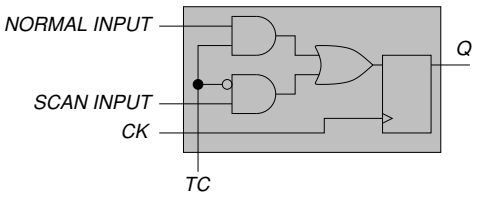


Redesign to avoid clock rule violation.

(ii) The following figure shows the scan wiring. All three D-flip-flops are replaced by scan flip-flops (SFFs) shown as shaded blocks. The logic of SFF, which includes a D-flip-flop and a multiplexer, is shown in the second figure.



Scan circuit (scan wiring shown in bold lines.)



Scan flip–flop (SFF).

Problem 8: BIST (12 Points)

1. For the circuit in Figure 6, please design an external-XOR LFSR pattern generator implementing the characteristic polynomial $1+x^2+x^3$ and an Input MUX for testing.

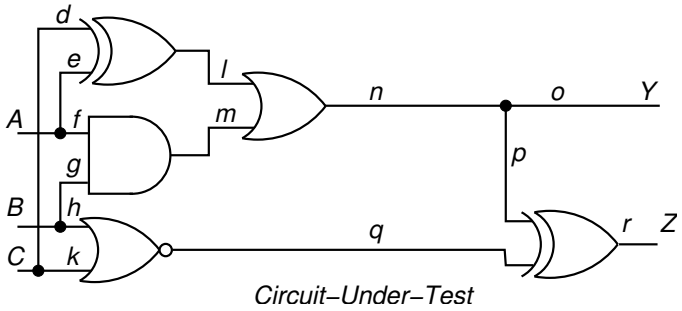


Figure 6: Circuit for Problem 8.

2. Express the linear system of matrix equations describing this pattern generator.

3. Now, assume an exhaustive counter-based pattern generator and the given response compacter for the circuit in Figure 7. Compute the good machine signature for the circuit. The output MISR is initialized to "000" before testing.

4. For the same circuit, compute the bad machine signature for the fault q stuck-at-0. Does the test hardware alias for this fault?

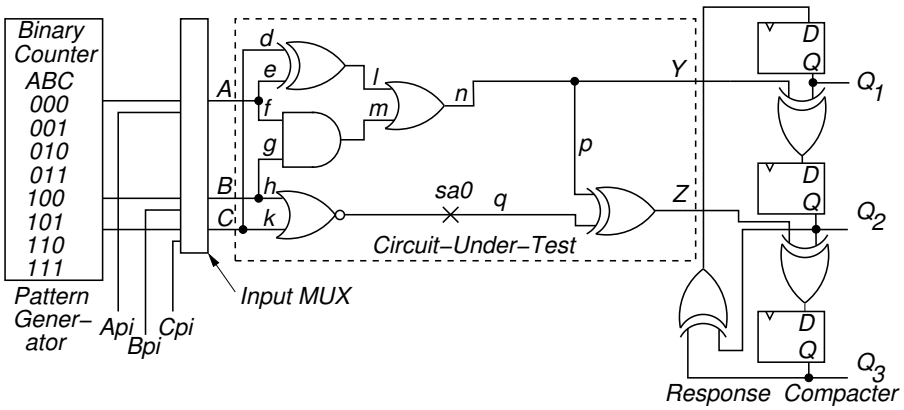


Figure 7: Circuit for Problem 8 with BIST hardware.

Solution to Problem 8

1. The hardware is shown in Figure 8.

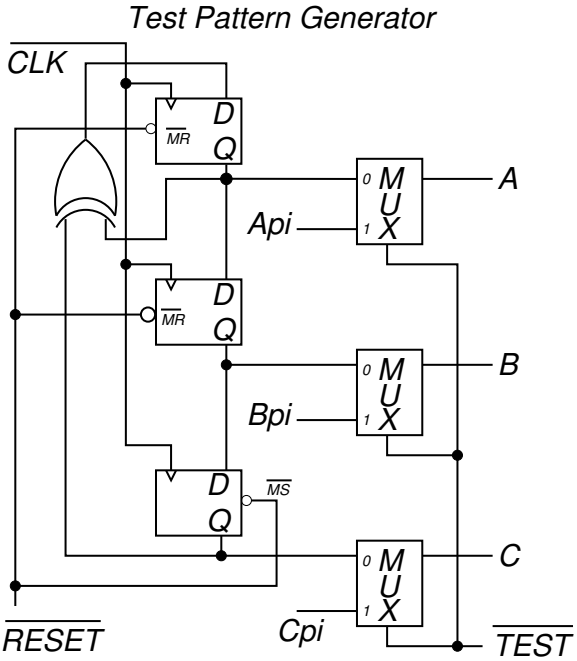


Figure 8: Circuit for Problem 8 with BIST pattern generator and input MUX.

2.

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix}(t+1) = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix}(t)$$

3. The table below contains the fault-free outputs of the circuit and the state of the LFSR after every clock. The initial state of the flip-flop is assumed to be $Q_1Q_2Q_3 = 000$. The output equations used for computing the fault-free outputs in the table are:

$$Y = (A \oplus B) + AB \quad \text{and} \quad Z = \overline{B + C} \oplus Y$$

A	B	C	Y	Z	LFSR state		
					Q_1	Q_2	Q_3
0	0	0	0	1	0	0	1
0	0	1	1	1	1	1	1
0	1	0	0	0	0	1	1
0	1	1	1	1	0	1	0
1	0	0	1	0	1	1	1
1	0	1	0	0	0	1	1
1	1	0	1	1	0	1	0
1	1	1	1	1	1	1	0

Thus the final signature of the good machine is "1 1 0".

The logic to detect this signature can be implemented by a NAND gate as evident from the following equation.

$$\overline{GOOD} = \overline{Q_1 Q_2 \overline{Q_3}}$$

4. In the case of the fault q s-a-0, the faulty outputs are:

$$Y_f = Y \quad \text{and} \quad Z_f = Y$$

The table below contains the faulty outputs of the circuit and the state of the LFSR after every clock. The initial state of the flip-flops is assumed to be $Q_1Q_2Q_3 = 000$ as before.

A	B	C	Y	Z	LFSR state		
					Q_1	Q_2	Q_3
0	0	0	0	0	0	0	0
0	0	1	1	1	0	1	1
0	1	0	0	0	0	0	1
0	1	1	1	1	1	1	1
1	0	0	1	1	0	0	0
1	0	1	0	0	0	0	0
1	1	0	1	1	0	1	1
1	1	1	1	1	0	1	0

Thus, the final signature of the faulty circuit will be "0 1 0", and the test hardware does not alias.

Problem 9: Boundary Scan (11 Points)

Figure 9 shows two boundary scan cells surrounding the on-chip system logic. We test the path from the INPUT boundary scan cell, through the on-chip system logic, and ending at the OUTPUT boundary scan cell. The JTAG commands are: *SAMPLE*, *PRELOAD*, *EXTEST*, *INTEST*, *RUNBIST*, *CLAMP*, *IDCODE*, *USERCODE*, *HIGHZ*, and *BYPASS*. Please explain the sequence of these commands used for delay fault testing of this particular path.

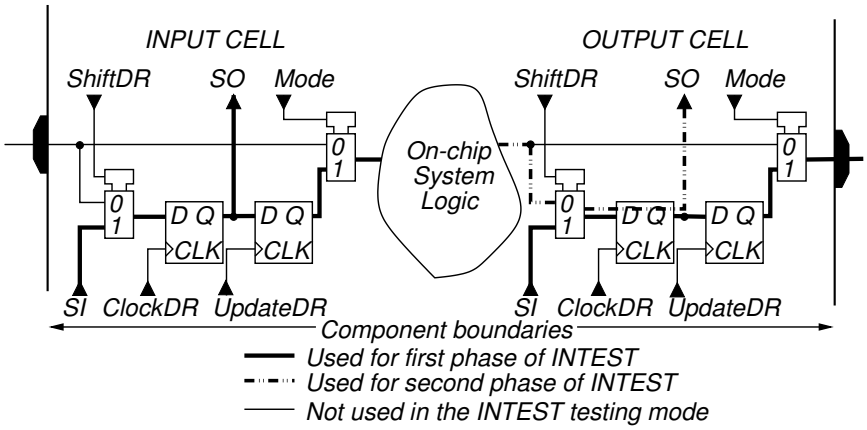


Figure 9: Circuit for Problem 9.

Solution to Problem 9

Here is the sequence of JTAG commands used to test the path:

1. SAMPLE – Capture pin signals and functional hardware outputs in the Boundary Scan Register.

2. Apply INTEST – Copy Boundary Scan Register contents into hold latch. Apply as many TCK pulses as are necessary to shift the 1st test pattern in from TDI (this will be the number of pins from TDI up to and including this system input pin.) Then apply a functional system clock.

3. Repeat Step 2 to apply the 1st time frame pattern to the path and shift the 2nd time frame pattern into the Boundary Scan Register (requires many TCK pulses.) At this point, the 1st time frame pattern is applied from the hold registers and the second time frame pattern is in the Boundary Scan Register.

4. Repeat the INTEST instruction and immediately apply a functional system clock. This applies the second time frame pattern to the path-under-test, clocks the functional circuit, and captures its response in the Boundary Scan Register. Then, apply as many TCK pulses as are necessary to shift out the response to the 2nd pattern out through TDO (while simultaneously shifting

in the next test pattern.) The shifted out response will indicate where a timing fault has occurred.

5. Go back to Step 2 if more paths are to be tested.

This sequence can be preceded by IDCODE and USERCODE instructions, in order to identify which chips, and which part programmings, are in the system. Other chips in the system may be given BYPASS instructions to speed up scan chain shifting by shortening the scan chain.

Final Exam: Graduate Course – VLSI Testing

Auburn Univ., ELEC 7250, Spring 2004

May 10, 2004

Instructions (please read before you proceed):

1. Please read all problems before starting your answers. Problems can be answered in any order.
2. Attempt all six problems and attempt all parts within each problem.
3. Answers can be written on question sheets or separate sheets or a combination. Each sheet should have a page number and problem number. On the first sheet write your name and the total number of sheets you are submitting.
4. Before handing in your answers, please check them thoroughly. If necessary, extra 10 minutes can be allowed for checking.

Problem 1: Fault Modeling (16 Points)

For the circuit of Figure 1, determine fault equivalences among the five faults shown.

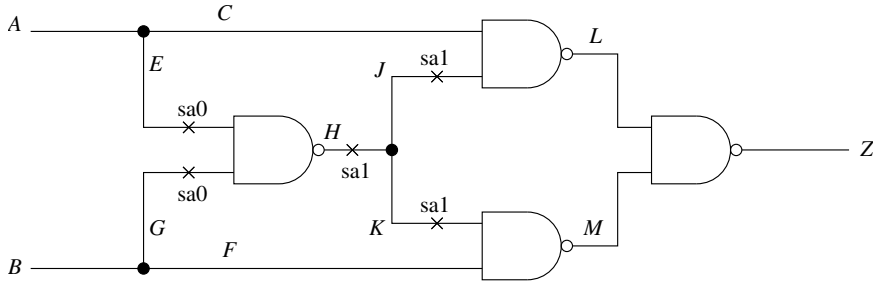


Figure 1: Circuit for fault modeling problem.

Solution to Problem 1

Three faults, E sa0, G sa0, and H sa1 are structurally equivalent as they are on the inputs and output of a NAND gate. The output function for any of these faults is $A + B$. The same output function is produced by J sa1 and K sa1 also. Therefore, all five faults are equivalent.

Problem 2: Testability Measures (16 Points)

For the circuit of Figure 2 compute SCOAP combinational controllability and observability measures for all lines (8 points).

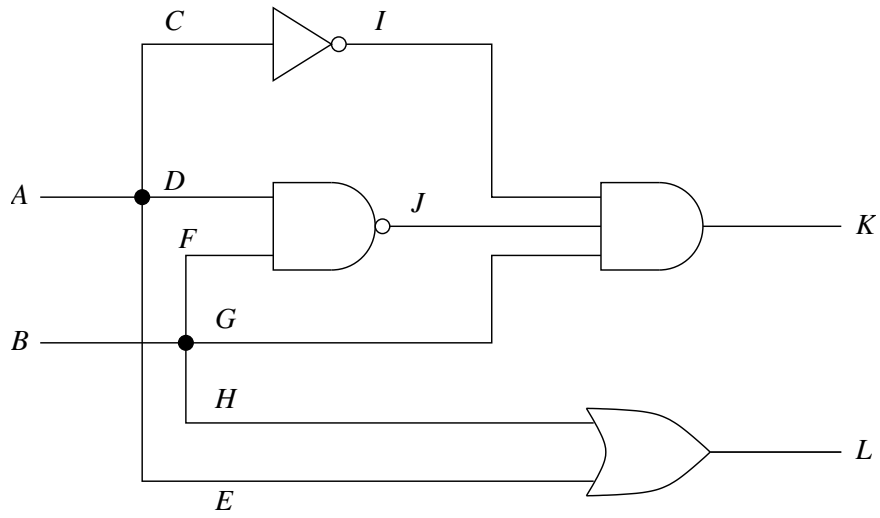


Figure 2: Circuit for testability measures problem.

(i) Assuming that the testability of a stuck-at fault can be represented as the sum of appropriate controllability and observability, find the set of most difficult to test faults (4 points).

(ii) Proving a fault to be redundant is a difficult task for an ATPG program. This circuit has three redundant faults, I s-a-1, J s-a-1, and F s-a-1. Are all of these faults in your set of most difficult to test faults? If not, explain why not (4 points).

Solution to Problem 2

Combinational SCOAP testsbility measures, $CC0$, $CC1$ and CO, are shown in Figure 3. For fanout lines only observabilities are shown because their controllabilities are the same as those of their stems.

(i) Most difficult to test faults in this circuit are those for which controllability and observability adds up to 7. There are five such faults: D sa0, D sa1, F sa0, F sa1, and J sa1. Since we can collapse the sa0 faults of the inputs of the NAND gate J into the output sa1 fault, this set reduces to three faults: D sa1, F sa1, and J sa1.

(ii) Two out of the three given redundant faults are in the set of difficult to test faults we identified. The third redundant fault, I sa1 has a SCOAP testability of 6 that is just below the maximum. To activate this fault, we must set $I = 0$ and $A = 1$, making F and J correlated as $J = \overline{F}$ because of the NAND gate. Since F and G have the same value, two inputs J and G of the AND gate K must assume opposite values. This makes the observation of the fault I sa1 impossible. While computing the observability of line I, SCOAP assumes

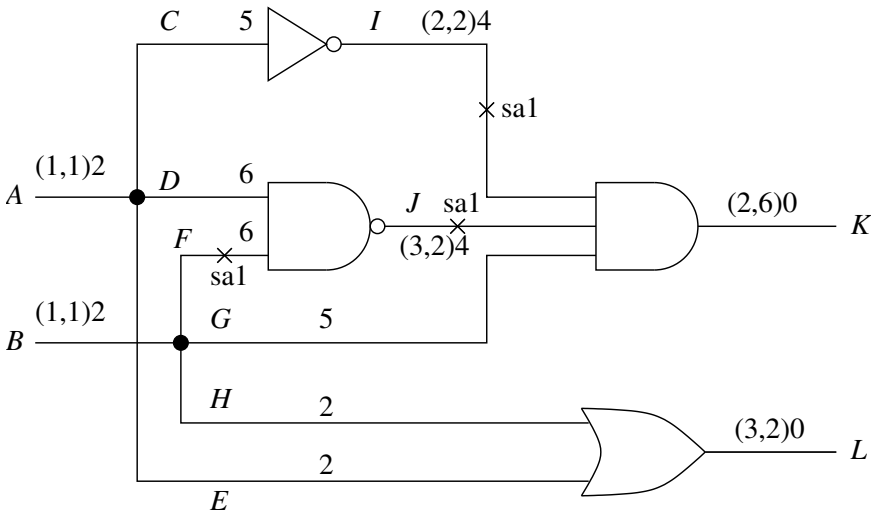


Figure 3: SCOAP measures, $(CC0, CC1)CO$, for the circuit of Figure 2.

J and G to be independently controllable and adds their $CC1$. Thus, the difficulty of testing the redundant fault I sa1 is underestimated.

Problem 3: Logic Simulation (17 Points)

The following conversation is recorded at a meeting to discuss the verification problems of the circuit shown in Figure 4:

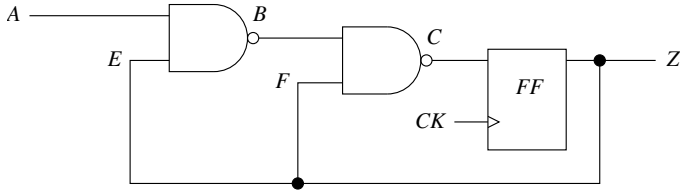


Figure 4: Circuit for Problem 3.

Designer: Why does our logic simulator not initialize my circuit?

CAD Engineer: That is due to a limitation of the three-state logic simulator.

Designer: Irrespective of the initial state, the flip-flop should initialize to 1 state when 1 is applied at the input A and the circuit is clocked. I am not going to change the circuit. You better fix your simulator.

CAD Manager (not realizing what is being promised): My engineer will find a solution before the end of the day.

You are the CAD engineer who *must* take the challenge:

(i) Devise a modification of the three-state logic simulator that will correctly initialize the circuit of Figure 4 containing a single flip-flop (9 points).

(ii) How will you extend the new procedure for circuits with many flip-flops (4 points)? Discuss any limitations (4 points).

Solution to Problem 3

(i) We add a fourth value to the three-value set $\{0, 1, X\}$. This value, denoted as $\overline{X}$, is the complement of the unknown or don't case state X. For a two-input NAND gate, the function is given by the table below:

Table 1: NAND function with four-state logic.

	0	1	X	$\overline{X}$
0	1	1	1	1
1	1	0	$\overline{X}$	X
X	1	$\overline{X}$	$\overline{X}$	1
$\overline{X}$	1	X	1	X

With this logic system, we start the simulation by setting $Z = X$ and apply $A = 1$ as the *designer* specified. Since $E = F = X$, the table gives, $B = \overline{X}$ and $C = 1$. After the clock CK is applied, we have $Z = 1$ and the flip-flop is initialized. Similar tables can be derived for all other logic gates and for gates with more than two inputs.

(ii) If there are many flip-flops that are initially in the unknown state, then each should be assigned a separate value, X_1, X_2, etc. Consider a NAND gate whose inputs have more than one X, say X_i and X_j. If $i = j$ then the output will be determined by the the above table. If $i \neq j$ then X's should be interpretted as ordinary X's of the three-valued logic system. Thus, the number of logic states will be $\{0, 1, X, X_1, X_2, \ldots, X_n, X\}$, when the circuit has n flip-flops.

Limitation: This method of simulation has much higher complexity when the circuit has a large number of flip-flops. This is because for n flip-flops, we must use an $n + 3$ valued logic. However, as a flip-flop get initialized, the corresponding X_i state disappears. Altenatively, any X_i state can be converted to X state, reducing the number of simulation states. The effect of such an action is to increase the pessimism in simulation.

Problem 4: Podem (17 points)

Use the Podem algorithm to derive a test for the fault G stuck-at-0 in the Schneider's example circuit from Roth's 1967 paper, which is shown in Figure 5 (8 points).

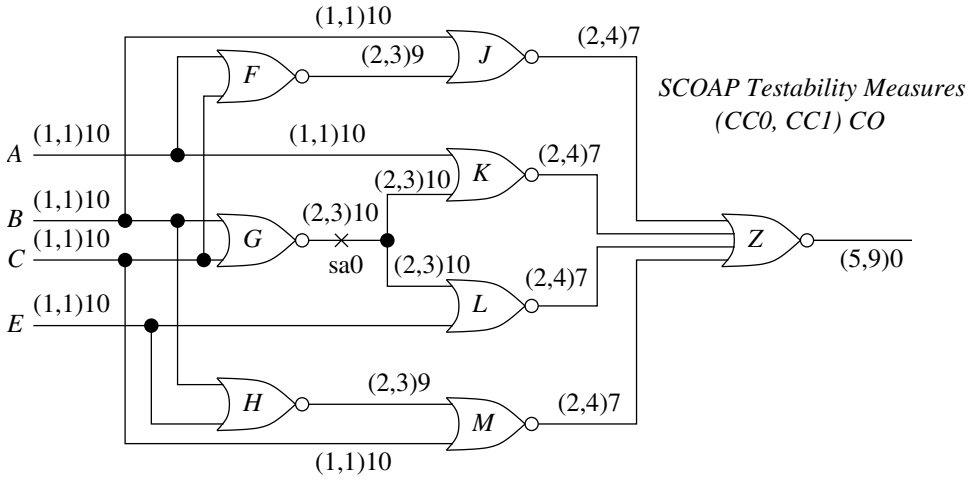


Figure 5: Schneider's circuits for Podem problem.

Neatly write all steps, specifying objectives, operations (backtrace, forward implications, X-path check, etc.) Give the state of the implication stack at each step (9 points).

Solution to Problem 4

The test for fault G sa0 is $A = 0,\ B = 0,\ C = 0,\ E = 0$. The steps of Podem are given in the following table.

Step No.	Objective	Action	Imp. stack	Implied signal values $A, B, C, E, F, G, H, J, K, L, M, Z$	D front.	X path
1	**Fault activation** $G = 1(D)$	Backtrace	$B = 0$	$B = 0$	ϕ	-
2	$G = 1(D)$	Backtrace	$C = 0$ $B = 0$	$B = 0, C = 0, G = 1(D)$	K, L	OK
3	D **propagation** $A = 0$	Backtrace	$A = 0$ $C = 0$ $B = 0$	$B = 0, C = 0, G = 1(D)$ $A = 0, F = 1, J = 0, K = \overline{D}$	L, Z	OK
4	$L = 0$	Backtrace	$E = 1$ $A = 0$ $C = 0$ $B = 0$	$B = 0, C = 0, G = 1(D)$ $A = 0, F = 1, J = 0, K = \overline{D}$ $L = 0, H = 0, M = 1, Z = 0$	ϕ	NONE
5	Restore X path	Backtrack	$E = 0$ $A = 0$ $C = 0$ $B = 0$	$B = 0, C = 0, G = 1(D)$ $A = 0, F = 1, J = 0, K = \overline{D}$ $L = \overline{D}, H = 1, M = 0, Z = D$	Fault detected	
Algorithm termination: Test $A = 0, B = 0, C = 0, E = 0$ detects the fault as $Z = D$ (1 backtrack)						

Problem 5: Delay Test (17 points)

(i) Specify a single input change (SIC) test for the critical path $\uparrow a - z$ (shown in bold lines) in the circuit of Figure 6(i). Is this a robust test? (8 points)

(ii) The circuit of Figure 6(i) is redesigned in Figure 6(ii) to reduce the delay. Will the SIC test obtained above still test the longest delay path shown in bold lines? If not, what is the minimum modification required in the test? (9 points)

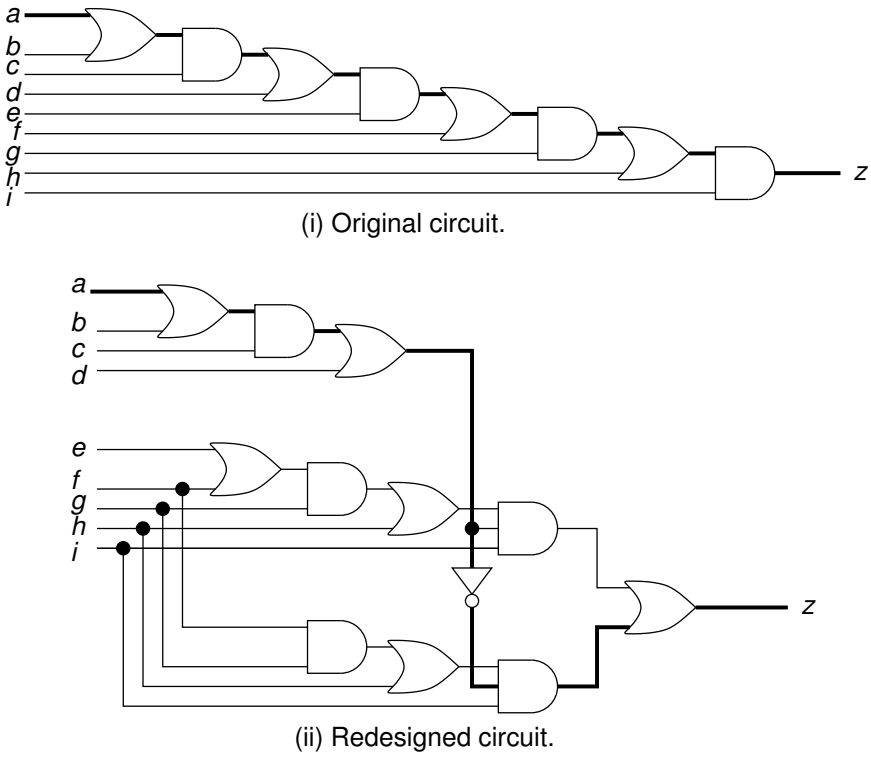


Figure 6: Circuits for delay test problem.

Solution to Problem 5

(i) A SIC test is found by statically sensitizing the path $a - z$ and applying a rising transition to a. Thus, the test is: $a = R1$, $b = S0$, $c = S1$, $d = S0$, $e = S1$, $f = S0$, $g = S1$, $h = S0$, $i = S1$. This is a robust test.

(ii) When the above test is applied to the redesigned circuit, the OR gate at the output receives a rising transition ($R1$) at its upper input and a $S0$ at the lower input. Although $R1$ is produced at Z, this transition does not arrive through the longest path (shown in bold). Thus, the upper path, which is 5-gate long,

is tested but the 6-gate path (shown as critical path) is not tested. We find that no input vector can sensitize this path when $a = 1$ and the PDF $\uparrow a - z$ for the critical path is untestable.

This part not required: When $a = 0$, setting $h = S1$ and leaving all other inputs as before, we can sensitize the critical path. Thus, the PDF $\downarrow a - z$ for the critical path can be tested by applying $a = F0$, $h = S1$ and leaving all other inputs as before. We also note that the s-a-1 fault at the output of the NOT gate is redundant. If that fault is removed, then the 6-gate critical path will also be removed. Then, the 5-gate path becomes the critical path and it is robustly testable by the test derived in part (i).

Problem 6: DFT (17 Points)

The circuit in Figure 7 is a sequence detector. A sequence 11111 in the input bit-stream at A locks the output Z to 1. The output can be unlocked by applying $R = 1$.

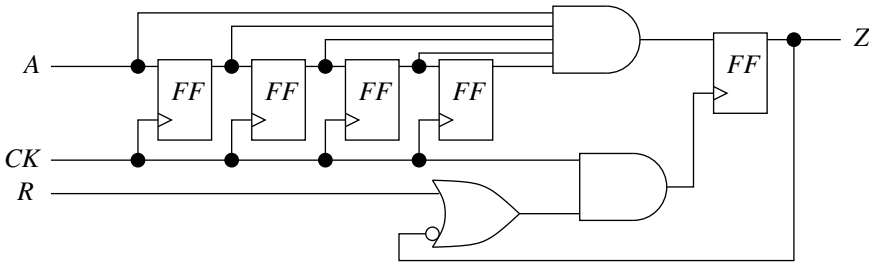


Figure 7: Circuit for DFT Problem 6.

For scan design, only one pin is available and the only circuit element that can be used is a two-to-one multiplexer.

(i) Redesign the circuit using minimum extra hardware to conform to the scan design rule, "clock must not be gated by a combinational signal." Neatly sketch the redesigned circuit (8 points).

(ii) Sketch a schematic of the full-scan circuit using minimum extra overhead. Show the complete wiring of the $SCANIN$, $SCANOUT$ and test control (TC) signals (9 points).

Solution to Problem 6

The redesigned scan circuit is shown in Figure 8. Two multiplexers have been added and one AND gate is eliminated. Only one extra pin is used for the test control signal TC. The added wiring is shown in bold lines.

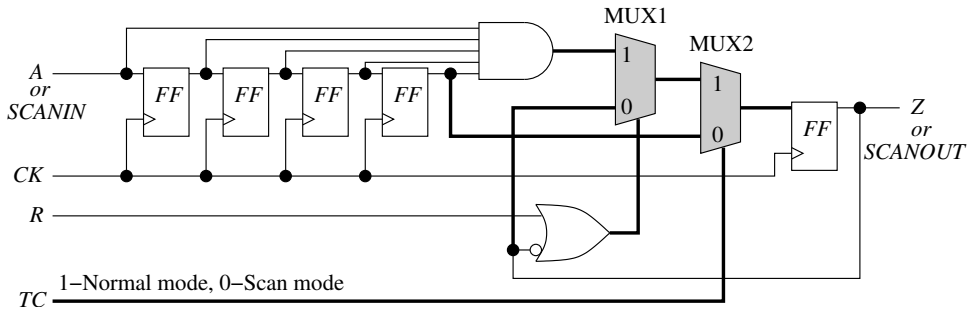


Figure 8: Scan design for the circuit of Figure 7.

(i) *Work-around for design rule violation:* The clock signal CK is directly supplied to the last flip-flop. The output of the OR gate now controls a multiplexer, MUX1. When the control is 0, which can only occur if Z is 1, Z is fed back into the last flip-flop so that its state remains unchanged. The control will be 1 if either $Z = 0$ or $R = 1$. In that case the output of the sequence detecting AND gate is supplied to the flip-flop. The logic of the multiplexer and the OR gate can be reduced but this design is based on the use of the given multiplexer block.

(ii) *Scan design:* Since the first four flip-flops already form a shift register, no scan modification is necessary. The last flip-flop is preceded by another multiplexer, MUX2, in addtion to the one inserted to eliminate the clock rule violation. The scan wiring is shown in bold lines. Input A is also used as $SCANIN$ and Z is used as $SCANOUT$. The test control signal TC is the only added input pin. In the normal mode ($TC = 1$), the circuit functions as sequence detector. In the scan mode ($TC = 0$), the output of the fourth flip-flop is routed into the last flip-flop, thus forming a shift register of length five between A and Z pins of the circuit.

Final Exam: Graduate Course – VLSI Testing

Auburn Univ., ELEC 7250, Spring 2005

April 30, 2005

Instructions (please read before you proceed):

1. Please read all problems before starting your answers. Problems can be answered in any order.
2. Attempt all five problems and attempt all parts within each problem.
3. Answers can be written on question sheets or separate sheets or a combination. Each sheet should have a page number and problem number. On the first sheet write your name and the total number of sheets you are submitting.
4. Before handing in your answers, please check them thoroughly. If necessary, extra 10 minutes can be allowed for checking.

Problem 1: Fault Modeling (20 Points)

(a) In a single-output circuit if two faults are indistinguishable, i.e., they produce the same output function, then prove that the faults have the same set of tests. (10 points)

(b) For the circuit of Figure 1, derive tests for faults i sa1 and e sa0. (5 points)

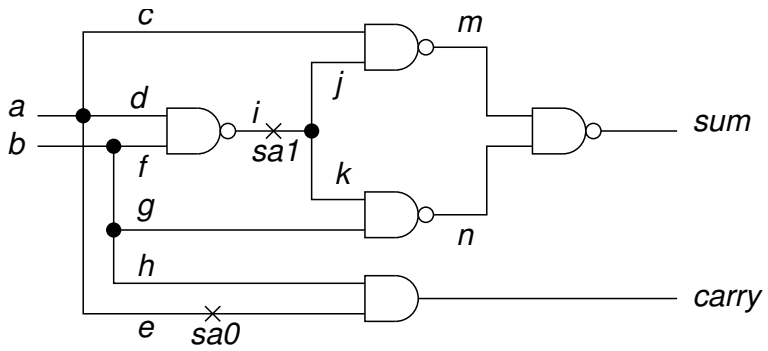


Figure 1: Circuit for Problem 1 on fault modeling.

(c) Are two faults in part (b) distinguishable? If yes, how will you distinguish between them? If not, why not? (5 points)

Solution to Problem 1

(a) Consider a Boolean vector X, a logic function $f(X)$, and two faults with corresponding functions $f_1(X)$ and $f_2(X)$. For the faults to be indistinguishable, the two faulty functions must assume identical values for all X:

$$f_1(X) \equiv f_2(X), \ \forall \ X \tag{1}$$

Assume that vector X_1 is a test for fault 1. Then,

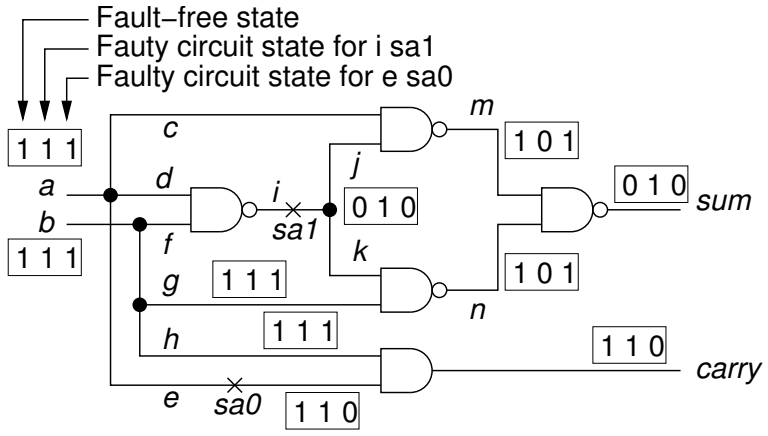
$$f_1(X_1) \oplus f(X_1) = 1 \tag{2}$$

From Equation (1) $f_1(X_1) \equiv f_2(X_1)$ and substituting this in Equation (2) we get

$$f_2(X_1) \oplus f(X_1) = 1 \tag{3}$$

which implies that X_1 is also a test for fault 2. Similarly, any test X_2 for fault 2 can be shown to be a test for fault 1. Thus, both faults have exactly the same tests. ■

(b) Both faults, i sa1 and e sa0, are detected by exactly one vector, $a = b = 1$. This vector is simulated for both faults in parallel in the following figure.



Parallel fault simulation of test $a = b = 1$ for circuit of Figure 1.

(c) Even though the two faults have the same test set, they are distinguishable because the two faulty functions differ at each output. For the faults to be indistinuishable, the condition of Equation (1) in part (a) must be satisfied at each output. The following table gives a diagnostic procedure:

Inputs		Outputs		Fault
a	b	sum	$carry$	diagnosis
1	1	0	1	No fault
1	1	1	1	Fault i sa1 is present
1	1	0	0	Fault e sa0 is present
1	1	1	0	Some other fault is present

Problem 2: Testability Measures (20 Points)

(a) For the asynchronous circuit of Figure 2 compute SCOAP combinational controllability ($CC0, CC1$) and combinational observability (CO) measures for all lines. (10 points)

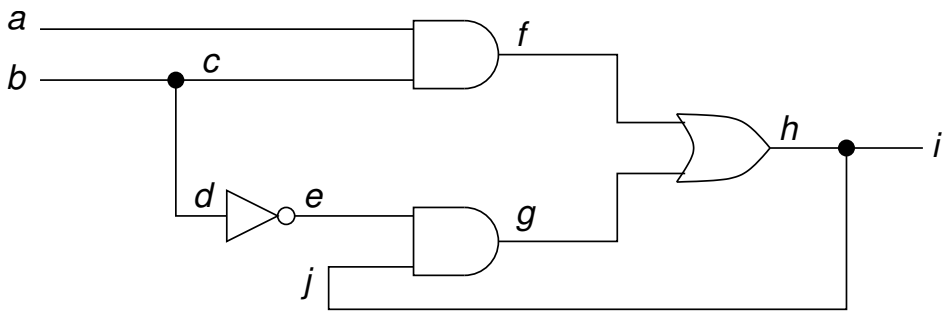


Figure 2: Circuit for Problem 2 on testability measures.

(b) Assuming that the testability of a fault can be represented as the sum of the appropriate controllability and observability (e.g., testability measure of a stuck-at-1 fault will be $CC0 + CO$), identify the most difficult to test stuck-at fault. (5 points)

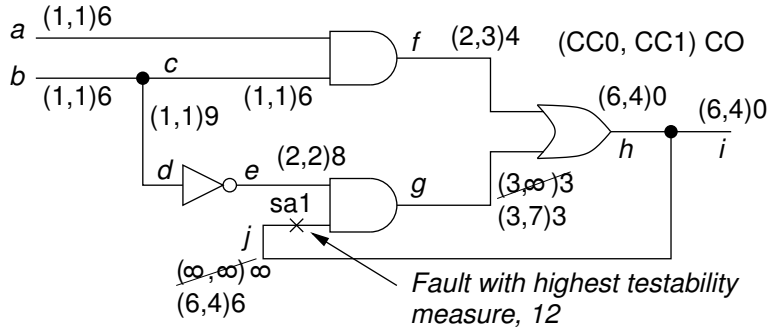
(c) Derive a test for the fault identified in part (b), considering line j to be in *unknown* state, initially. (5 points)

Solution to Problem 2

(a) Combinational SCOAP testsbility measures, $CC0$, $CC1$ and CO, are shown in the following figure. Initially, line j is set to $CC0 = CC1 = CO = \infty$. All values stabilize after two iterations.

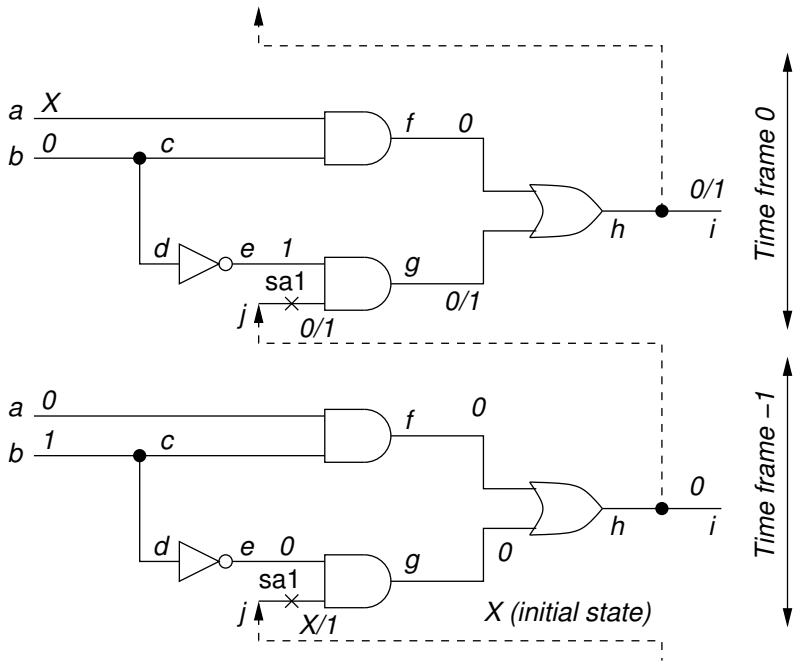
(b) The highest testability measure is $CC0 + CO = 6 + 6 = 12$ for fault j sa1.

(c) A test for the fault j sa1 consists of two vectors, (a, b) = (0,1) $\rightarrow$ (X,0). Derivation of this test by time-frame expansion is illustrated in the following



SCOAP measures, $(CC0, CC1)CO$, for the circuit of Figure 2.

figure. Asynchronous feedback from h to j is broken and initialization $j = X$ is used. The stability of signals in each time-frame is verified by applying the new state of h into j.



Test generation for fault j sa1 in the circuit of Figure 2.

Problem 3: ATPG and fault simulation (20 Points)

Derive a pseudo-combinational circuit for the cycle-free sequential circuit of Figure 3 by shorting all flip-flops (FF). Obtain a combinational test for the fault g sa1 in the pseudo-combinational circuit. If you succeed in finding a combinational vector then using fault simulation of the sequential circuit show that the targeted fault is detected by repeating the same vector. Otherwise, use the time-frame expansion method to derive a test sequence for this fault. (20 points)

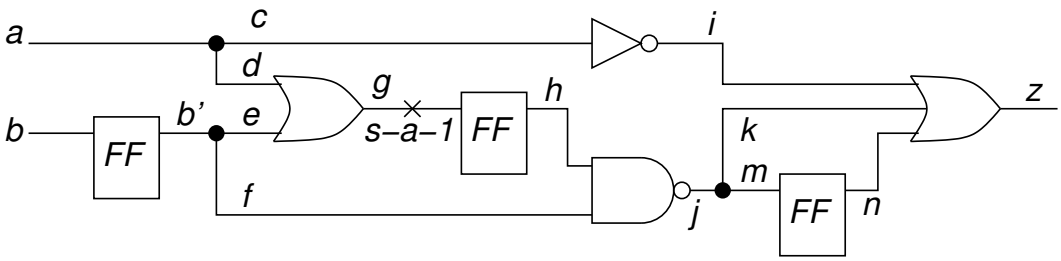
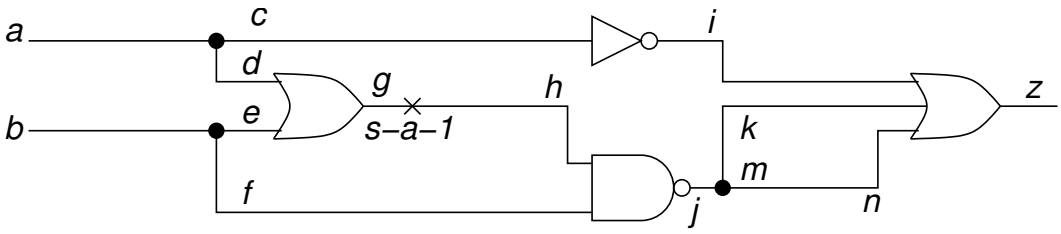


Figure 3: Circuit for Problem 3.

Solution to Problem 3

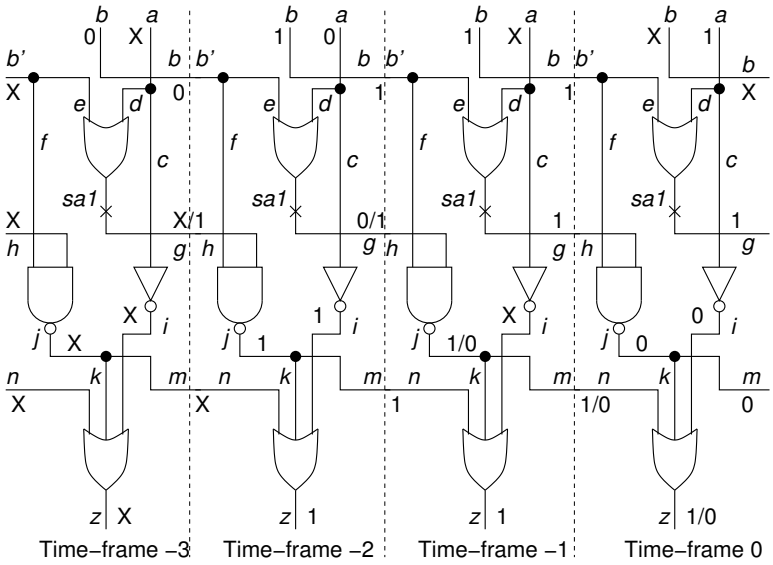
The pseudo-combinational circuit for the circuit of Figure 3 is shown below:



Pseudo-combinational circuit for sequential circuit of Figure 3.

A test for g sa1 in the pseudo-combinational circuit is impossible because $a = b = 0$ input that is essential to activate the fault, blocks the observation of the fault effect through the NAND gate j.

This fault is testable in the sequential circuit of Figure 3 because the activation of the fault and propagation through j can take place in separate time-frames. A valid test consists of four vectors, (a, b) = (X,0), (0,1), (X,1), (1,X). Its derivation is shown in the following figure:



Test generation for g sa1 fault in the sequential circuit of Figure 3.

Problem 4: Fault diagnosis (20 points)

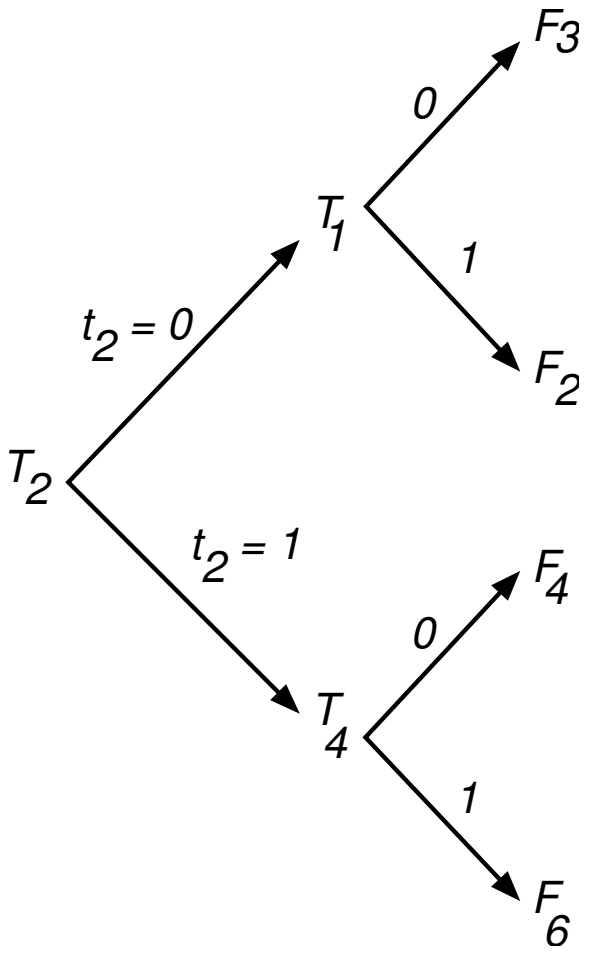
A circuit has six faults that are tested by four tests. Table 2 shows the fault dictionary, where $t_i = 1$ for fault Fj if ith test detects Fj. Faults $F1$ and $F5$ occur with negligible probability and all other faults are found to be equally likely in a faulty circuit. Devise a diagnostic tree to minimize the average time of diagnosing a faulty circuit. Given a faulty circuit, on an average how many tests will be applied before the fault is found. (20 points)

Table 2: Fault dictionary for Problem 4.

Fault	Occurrence probability	Test syndrome			
		t_1	t_2	t_3	t_4
No fault	-	0	0	0	0
$F1$	0.00	0	0	0	1
$F2$	0.25	1	0	0	0
$F3$	0.25	0	0	1	0
$F4$	0.25	0	1	0	0
$F5$	0.00	1	0	1	0
$F6$	0.25	0	1	0	1

Solution to Problem 4

Let us denote the tests as T_1, T_2, T_3 and T_4, such that t_i is the test syndrome of T_i. We will only consider the four faults that have non-zero probabilities of occurrence. A binary search generally leads to an optimum diagnosis since each test provides a pass/fail result. Examining the tests of Table 2, we find that application of T_2 divides the fault set into subsets (F_4, F_6) and (F_2, F_3). Similar consideration leads to the following diagnostic tree. We have assumed that the circuit is known to be



faulty before the diagnosis begins. Time for diagnosis is estimated as follows:

Average diagnostic test length

$$
\begin{aligned}
&= 2 \times Prob(F_4) + 2 \times Prob(F_6) + 2 \times Prob(F_2) + 2 \times Prob(F_3) \\
&= 2 \times 0.25 + 2 \times 0.25 + 2 \times 0.25 + 2 \times 0.25 \\
&= 2.0
\end{aligned}
$$

The average diagnosis time is two times that of applying one test.

Problem 5: Scan design (20 Points)

The circuit in Figure 4 is a sequence detector. Only one 100-bit long sequence 0 . . . 11010 in the input bit-stream at PI produces an output $PO = 1$. The 100-input AND function is implemented as a tree network of AND gates. The circuit is customized for a specific bit-stream by inverting the selected inputs of the AND network.

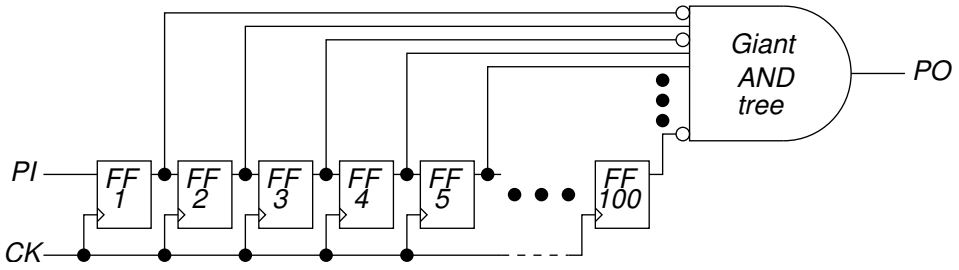


Figure 4: Circuit for scan design Problem 5.

(a) Considering the AND function as a stand-alone combinational circuit, what is the minimum number of combinational vectors to completely test all single stuck-at faults? (5 points)

(b) Implement scan design with minimum hardware and test pin overheads. What is the test length in clock cycles for testing the combinational logic and the shift register? (10 points)

(c) Modify the design to reduce the test time by a factor of 10. Add minimal hardware and no more than 20 pins to the original circuit. (5 points)

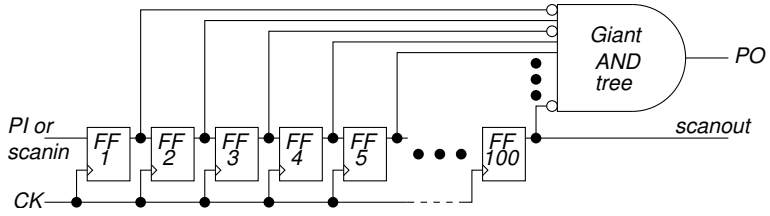
Solution to Problem 5

(a) The AND gate will require 101 combinational vectors:

- A single all-1 vector will detect all 100 s-a-0 faults.
- A single s-a-1 fault on an input line i will be tested by a vector containing $i = 0$ and all other inputs set to 1. Since there are 100 such faults, 100 vectors will be needed.
- Inverters on input line will require the corresponding bit of the vector to be complemented. Because of the equivalence of their input and output faults, inverters do not require any additional tests.
- Tests for input stuck-at faults will cover all other faults in the AND tree. This is according to the checkpoint theorem on fault dominance.

(b) The minimal scan design requires only one extra pin that is used as the *scanout* signal that is necessary for testing the shift register. PI would be used as

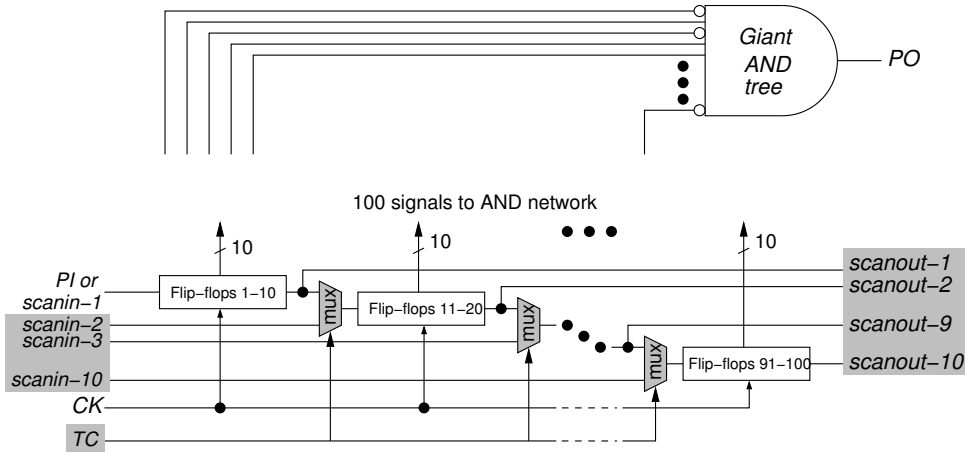
scanin. Because *scanin* and *scanout* are always available, no test control signal is required. The circuit is shown in the following figure:



Minimum overhead scan design of the circuit of Figure 4.
Only one pin for scan-out is added.

The shift register test will require $N_{FF} + 4 = 104$ clock cycles. For combinational tests of the AND gate, as soon as a vector is scanned in, the output of the AND gate is directly observed. So, no scan-out is needed. Thus, application of 101 combinational vectors will take $101 \times 100 = 10,100$ clocks. The total length of the scan test is $104 + 10,100 = 10,204$ clock cycles.

(c) To reduce the test time, nine scan multiplexers are inserted to break up the shift register into 10 segments, each containing 10 flip-flops. All multiplexers are controlled by a test control signal TC. The PI signal is used as the scan-in signal, *scanin-1*, for the first shift register segment. The other nine segments are supplied the scan-in through test pins, *scanin-2* through *scanin-10*. Each shift register segment has a test output pin. The scan schematic is shown in the following figure:



Reduced test time scan design of the circuit of Figure 4.
Hardware overhead is shown as grey shaded area.

The scan overhead consists of nine multiplexers and 20 pins (TC, 9 scanins, and 10 scanouts).

All shift register segments are tested in parallel by a sequence of 14 clocks. Scanning in of each combinational vector takes 10 clocks and, therefore, 1,010 clocks will be needed to apply 101 vectors. Thus, total test time is $14+1,010 = 1,024$ clock cycles.
